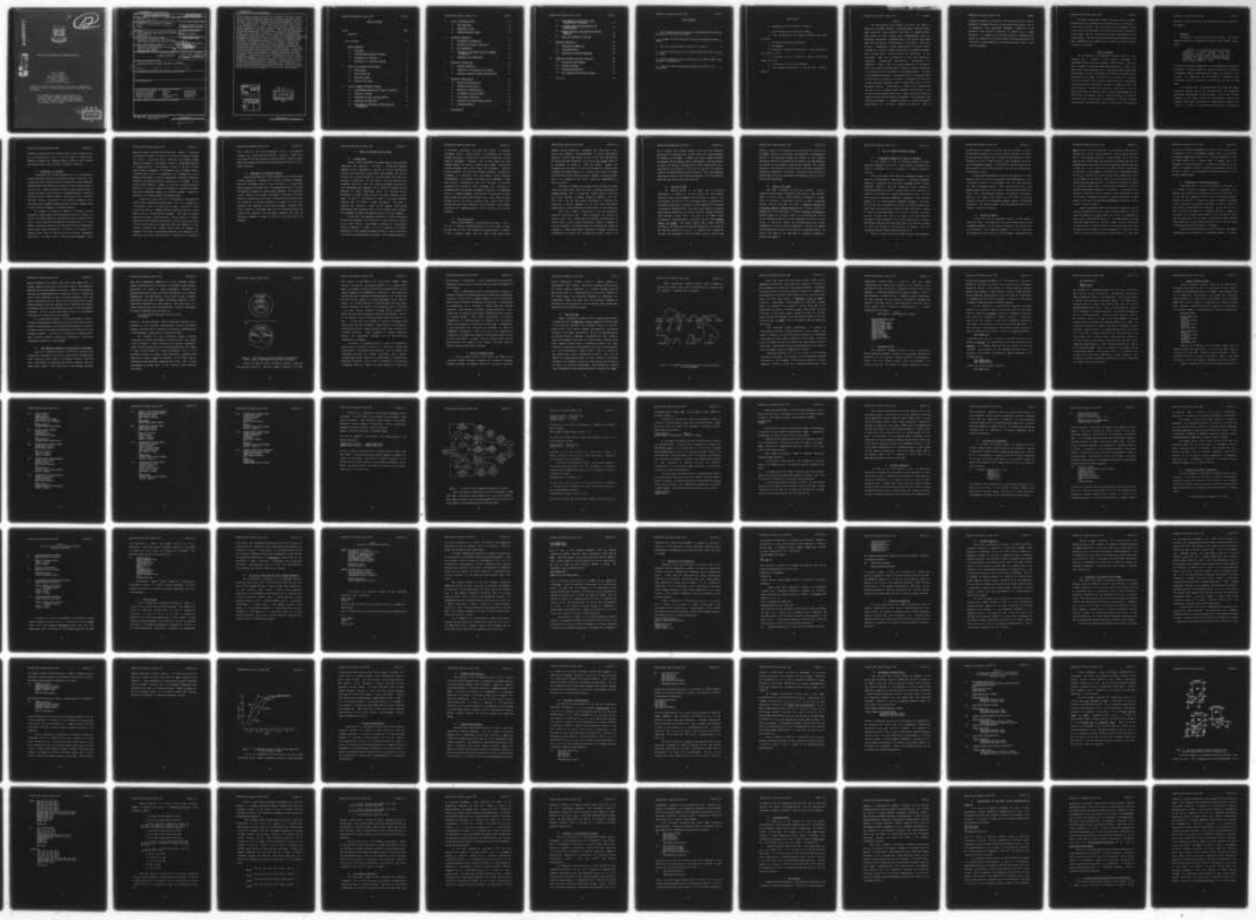
AD-A052 671 YALE UNIV NEW HAVEN CT DEPT OF PSYCHOLOGY F/G 5/10
A THEORY OF THE ACQUISITION OF COGNITIVE SKILLS.(U)
FEB 78 J R ANDERSON , P J KLINE, C M BEASLEY N00014-77-C-0242
UNCLASSIFIED NL

1 OF 2
AD
A052671



A Theory of the Acquisition of Cognitive Skills[1]

John R. Anderson
Paul J. Kline
Charles M. Beasley, Jr.
Department of Psychology
Yale University
New Haven, Connecticut 06520

This research was sponsered by the Personnel and Training Research programs, Psychological Services Division, Office of Naval Research, under Contract No.: N00014-77-C-0242, Contract Authority Identification Number, NR No. 154-399.

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: Technical report 77-1
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): (6) A theory of the acquisition of cognitive skills.
5. TYPE OF REPORT & PERIOD COVERED: Technical report 1/77-12/77
6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): (10) John R. Anderson, Paul J. Kline and Charles M. Beasley, Jr
8. CONTRACT OR GRANT NUMBER(s): (15) N00014-77-C-0242

9. PERFORMING ORGANIZATION NAME AND ADDRESS:
Department of Psychology - Yale University
Box 11a Yale Station - New Haven, CT 06520
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:
66153N
(17) RR042 04 01 NR 154-399

11. CONTROLLING OFFICE NAME AND ADDRESS:
Personnel and Training Research Programs
Office of Naval Research
Arlington, VA 22217
(11)
12. REPORT DATE: (8) Feb [redacted] 78
13. NUMBER OF PAGES:

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
(16) RR042 04
15. SECURITY CLASS. (of this report): unclassified (12) 136p.
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report):
Approved for public release; distribution unlimited.
(9) Technical rept. Jan-Dec 77

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Artificial intelligence	Learning	Strengthening
Associative networks	Memory	Generalization
Cognitive Psychology	Production system	Discrimination
Computer simulation	Designation	Composition

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

unclassified
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

402 628

The paper describes the ACT theory of learning. The theory is embodied as a computer simulation program that makes predictions about human learning of various cognitive skills such as language fluency, study skills for social science texts, problem-solving skills in mathematics, and computer programming skills. The learning takes place within the ACT theory of the performance of such skills. This theory involves a propositional network representation of general factual knowledge and a production system representation of procedural knowledge. Skill learning mainly involves addition and modification of the productions. There are five mechanisms by which this takes place: designation, strengthening, generalization, discrimination, and composition. Each of these five learning mechanisms is discussed in detail and related to available data in procedural learning. Designation is the process by which one production can designate another. The power of the designating production is postulated to vary with sophistication of the learner in the domain to be learned. Strengthening is the process by which successful productions gradually acquite more control over the processing resources. This mechanism is related to the available data about how a skill gradually becomes automatic. Generalization is the process by which productions extend their range of application beyond the domain for which they were originally designated. This mechanism nicely accounts for phenomena in language acquisition and concept development. Discrimination is a coercive mechanism by which the range of overgeneral productions is restricted. This mechanism is used to explain phenomena in language acquisition and how problem-solving skills, such as in mathematics, evolve specialized submethods. Composition is the process by which multiple productions can combine into a single production. It explains the Einstellung effects in problem-solving. Finally, we discuss the difficulties we have encountered working with the current ACT learning systems and the projected changes in ACT to deal with these problems.

1473B

Table of Contents

Section Page

Subsection

List of Figures iv

1. What is Learning? 3

1.1 Definition 4

1.2 Accretion Restructuring and Tuning 5

1.3 Instruction vs. Induction 7

1.4 Declarative vs. Procedural Learning 9

2. Goals of the Project and this Paper 10

2.1 Project Goals 10

2.2 ACT vis-a-vis LAS 11

2.3 Overview of Paper 13

2.4 Status of this Paper 14

3. ACT as a Computer Simulation Program 15

3.1 A Programming Language and a Theory of Cognition . . . 15

3.2 Theories vs. Models 16

3.3 Comparisons to Older Learning Theories 18

3.4 Optimality and Efficiency 19

3.5 Good Cognitive Psychology is Good Artificial Intelligence 21

4. ACT as a Performance Model 25

4.1 ACT's Data Base 26

4.2 Productions in ACT 29

4.3 Example Production System2 32

5. Production Designation 43

5.1 The Notation of Designation 44

5.2 Encoding of Procedural Instructions 46

5.3 The Preprocessor 50

5.4 The Initial Preprocessor and First Language Acquisition 51

5.5 Designation with Substitution 55

6. Production Strengthening 57

6.1 Strength Computation 58

6.2 Designation has Precedence over Strength 59

6.3 Interaction Between Strength and Specificity 61

7. Production Generalization 66

7.1 Designated Generalizations 67

7.2 Automatic Generalization 67

7.3 Definition of Generalization 68

7.4 Two Examples of Generalization 71

7.5 The Problem of Efficiency 80

7.6 Focusing of the Generalization Process 82

7.7 Overgeneralization 84

8. Discrimination 84

8.1 Discrimination by restriction versus discrimination by exception 86

8.2 An Example Requiring Discrimination and Generalization 87

8.3 Example Continued: Discrimination by Variant Spawning 92

8.4 Effect of Punishment on Learning 96

9. Production Composition 98

9.1 Definition of Composition 98

9.2 Einstellung Effect 100

9.3 Conditions for Evoking Composition 105

10. Significant Problems and Future Directions 106

10.1 Difficulties with Matching 107

10.2 Data-Flow Matching 110

10.3 Knowledge Representation 115

10.4 Final Remarks about Projected Changes 121

11. References 125

List of Figures

1. Venn Diagrams illustrating possible relationships between cognitive psychology and artificial intelligence. 23

2. A fragment of ACT's propositional network surrounding the concept @CANDY. 27

3. The flow of control among the productions in Table 2. 39

4. Increase with practice in ACT's correct application of the articles a and the. 65

5. The three examples of chairs presented to the program--adapted from Hayes-Roth & McDermott-1976. 74

6. A data-flow network combining overlapping conditions in five productions. 111

Table Captions

1. Linearization of the information in Figure 2.

2. A set of productions for adding two numbers.

3. A set of productions for encoding information about LISP structures.

4. Two initial preprocessing productions.

5. Give example.

6. Productions designated and generalized in the induction of the concept of a chair.

7. A production set for learning main clause structure plus number inflection.

8. A production set for water jug problems.

9. The productions implemented in the data flow network of Figure 6.

Abstract

The paper describes the ACT theory of learning. The theory is embodied as a computer simulation program that makes predictions about human learning of various cognitive skills such as language fluency, study skills for social science texts, problem-solving skills in mathematics, and computer programming skills. The learning takes place within the ACT theory of the performance of such skills. This theory involves a propositional network representation of general factual knowledge and a production system representation of procedural knowledge. Skill learning mainly involves addition and modification of the productions. There are five mechanisms by which this takes place: designation, strengthening, generalization, discrimination, and composition. Each of these five learning mechanisms is discussed in detail and related to available data in procedural learning. Designation is the process by which one production can designate another. The power of the designating production is postulated to vary with sophistication of the learner in the domain to be learned. Strengthening is the process by which successful productions gradually acquire more control over the processing resources. This mechanism is related to the available data about how a skill gradually becomes automatic. Generalization is the process by which productions extend their range of application beyond the domain for which they were originally designated. This mechanism nicely accounts for phenomena in language acquisition and concept development. Discrimination is a corrective mechanism by which the range of

overgeneral productions is restricted. This mechanism is used to explain phenomena in language acquisition and how problem-solving skills, such as in mathematics, evolve specialized submethods. Composition is the process by which multiple productions can combine into a single production. It explains the Einstellung effects in problem-solving. Finally, we discuss the difficulties we have encountered working with the current ACT learning systems and the projected changes in ACT to deal with these problems.

This paper describes the theory of learning that we have been developing within the framework of the broader ACT theory. The paper is divided into ten sections. The first three provide the background to the current work and its philosophy. The fourth section gives an overview of the ACT theory. The next five sections describe various aspects of the learning theory. The last section describes the future prospects of this model. Depending on your dispositions as a reader, you may want to start at Section 4 or Section 5.

1 What is Learning?

We are interested in understanding learning. This is a much neglected area of research in recent cognitive psychology. Our methodology has been to develop a computer simulation model, called ACT, which is capable of learning the same cognitive skills as a human. This model leads to predictions which can be tested against both existing data and data that we collect. The results of this empirical testing lead to improvements in the model. The function of this paper is to set forth the initial structure of our learning model and indicate the beginning contact that we have made between the model and empirical data. But before doing this we would like to provide a brief review of past and current perceptions of learning in order to provide some perspective from which to view our research. We will consider a number of concepts and distinctions that have been used in discussions of learning. Although these concepts and distinctions point in the direction of interesting

questions, we will find that they are not sufficiently sharp to guide our own research.

1.1 Definition

Learning is a notoriously difficult term to define. The classic behavioral definition in psychology is like the one in Hilgard & Bower (1966):

> Learning is the process by which an activity originates or is changed through reacting to an encountered situation, provided that the characteristics of the change in activity cannot be explained on the basis of native response tendencies, maturation, or temporary states of the organism (e.g., fatigue, drugs, etc.). (p.2)

This definition has many inadequacies as is acknowledged in such textbook discussions. The basic problem is to rigorously apply such a definition to diagnosing whether various behavioral changes are learning or non-learning. For instance, by the above definition, forgetting due to interference would be classified as learning--and this does not seem right.

The problem is that our judgments about what behavioral changes constitute learning rely on our intuitions about the theoretical mechanisms underlying the behavioral changes. Thus it does not seem possible to have a purely behavioral, non-theoretical definition of learning. The problem of definition is somewhat more tractable in the context of a theory of the mechanisms that underlie the behavioral

changes. Certain processes in the theory can be legislated as learning processes while others can be legislated as non-learning. The output of the learning processes is, naturally enough, learning and the output of the non-learning process is not learning. However, there are still a number of potential problems with such a definition of learning. First, of course, it is only as good as the theory and will become obsolete when the theory does. Second, it may not be possible to diagnose a particular behavior as learning or non-learning. It is typical of current cognitive theories that there are many mechanisms that could be underlying a single phenomenon. This richness of mechanism is demanded by the richness of human cognition, but it makes it hard to decide what processes are operating at any point. A third problem with a theoretical definition is that it may not correspond well with our pre-theoretical intuitions about what is learning. For instance, we may have the strong intuition that sometimes the action of a theoretical mechanism is to produce learning and sometimes it does not. That is, the theory's division of behavior into basic mechanisms may not line up with our intuitive division of behavior into learning and non-learning.

1.2 Accretion Restructuring and Tuning

The difficulties with both the behavioral and theoretical definitions of learning suggest the futility of trying to develop a theory of learning that treats learning as a unified concept. This is reinforced by the growing realization in cognitive psychology that there

are some important distinctions to be made among phenomena that are sometimes lumped together under the category of learning. Rumelhart & Norman (1976) have made distinctions between three modes of learning that they call accretion, tuning, and restructuring. Accretion they equate with learning of facts. It is most like the kind of learning studied intensively during the past two decades under the rubrics of "verbal learning" and "human memory." It can be thought of as writing data into a memory without implying any structural changes in the memory system. Relative to the other two kinds of learning, accretion is a rapid process. One can acquire permanently a new fact after less than a minute's study.

The processes of restructuring and tuning occur over longer periods of time than does accretion. They affect the creation and transformation of schemata. Schemata are the structures postulated by Rumelhart and Norman to be responsible for interpreting new information and storing these interpretations in memory. That is, schemata are the structures responsible for accretion. So, in part, restructuring and tuning are concerned with phenomena that would have been called "learning to learn" in an earlier era in psychology.

Restructuring refers to the creation of new schemata to deal with new phenomena. Presumably, language acquisition would be an example, par excellence, of restructuring in that it involves the acquisition of the ability to interpret previously uninterpretable word strings. Tuning is the process by which a schema is changed to make it perform better. An

instance of tuning would be the process by which a child adjusts his rule for 's pluralization so that it will only apply to regular nouns. Rumelhart and Norman take a somewhat polemical stance and refer to tuning and restructuring as "real learning" in contrast to accretion.

1.3 Instruction vs. Induction

A distinction that has been important to us is the distinction between learning by discovery versus learning through instruction. This is similar to, but not identical with, the distinction drawn by Rumelhart & Norman between restructuring and tuning on one hand versus accretion on the other hand. Learning by discovery refers to those situations where a concept or procedure must be induced from examples whereas learning by instruction refers to those situations where what is to be learned is described. The best illustration of this contrast is between a child learning his first language by observing others use their language versus an adult learning a language in a classroom situation.

The problem of learning by induction or discovery has been subjected to formal analysis and has been shown to be very difficult. A review of the formal analyses of this problem is given in Anderson (1976, Ch.12). We will just state the significant conclusions here. It is not possible in practical time limits to solve many induction problems which popular opinion asserts humans solve. For instance, it is popular (e.g., Chomsky & Miller, 1963) to think that the class of natural languages is equivalent to the formal class of context-sensitive languages or even

some other larger, less-restrictive formal class. However, no algorithm exists which can acquire in practical time limits an arbitrary language out of such a large class given an arbitrary (for example, random) sequence of examples of the language. There are overwhelming obstacles to language learning. These obstacles include (1) the size of these formal classes of languages; (2) the apparent lack of adequate negative information about what is not permitted in the language; and (3) the ambiguity and synonymy of natural language which potentially complicate enormously the meaning-to-sentence structure of the language. Of the various human behaviors, language is the best understood formally; therefore, it is unclear whether other aspects of cognitive development pose similar "impossible" demands on an induction system. However, the existence of one instance should be disturbing enough.

There are two ways to get out of the paradox of proposing that children induce impossible-to-induce things. One is to reassess one's interpretation of what it is they have to induce. We think it is the case that the class of languages humans are capable of learning is much smaller than all context-sensitive languages. This class is strongly constrained by their induction methods. We also believe in a second explanation of the induction paradox which is that the learning situation is not as unstructured as one might think, that the information a child receives is anything but a random selection from the language. The language a child hears is considerably simplified, very responsive to his needs, and does contain some very direct instruction as to linguistic

rules. Basically, the child's predicament is not a pure induction situation but a mixed instruction-induction situation. Indeed, while pure instruction and pure induction situations exist as abstract ideals, almost every interesting real-world learning situation turns out to be a mixture of the two.

1.4 Declarative vs. Procedural Learning

Another distinction that has been important to us is that between learning declarative knowledge and learning procedural knowledge. Declarative knowledge refers to facts like those in a history text whereas procedural knowledge refers to knowledge about how to do things like speaking a language or solving algebra problems. This distinction is also similar to that between the products of accretion versus the products of restructuring and tuning. We have been quite polemical about the difficulty of learning procedures and claimed "it has no real parallel in the acquisition of declarative knowledge." (Anderson, 1976, p.490). However, it is becoming clear that any interesting learning task involves both procedural and declarative components and that the declarative component can pose as difficult a learning task as the procedural.

2 Goals of the Project and this Paper

2.1 Project Goals

We would like to understand how humans improve their cognitive capabilities with experience. We think a particularly important component of this improvement is learning, the kind of learning that involves restructuring and tuning rather than accretion, the kind of learning that involves learning by discovery rather than the direct encoding of instruction, and the kind of learning concerned with procedures rather than facts. The preceding discussion was concerned with acknowledging the vagueness of the concept of learning and the vagueness of these distinctions within the learning concept. Nonetheless, we remain firmly convinced that these concepts, even if vague, do point in the direction of some of the most interesting questions about the nature of human intelligence. This paper is concerned with providing a report of the progress made towards understanding these questions within the ACT framework and the impact that attempting to answer these questions has had on the ACT framework.

We feel that first language acquisition by children constitutes the empirical phenomenon that provides the closest instance of what we want to study. However, child language acquisition is a particularly difficult phenomenon to study. It is not possible to do careful experiments on child language development because of ethical constraints and children's uncooperativeness as subjects. Child language acquisition

is intertwined inextricably with many other aspects of cognitive development. Thus, it would not seem to be reasonable to try to model language acquisition in isolation--one would have to model all of child cognitive development. Therefore, while we think about the child language problem, our research is concerned with a number of interesting learning problems faced by both adults and adolescents. With this subject population, we do not have the same problem of extracting the development of a particular skill from massive overall cognitive transformations. It is also the case that we have a commitment to make some practical contributions to this area. Therefore, we have been considering the learning of second languages, the acquisition of mathematics skills like algebra and geometry, acquisition of programming skills in a computer language like LISP, and the acquisition of study skills relevant to reading social science texts. While these particular phenomena do not turn out to provide us pure cases of the kind of learning that we are most interested in, they nonetheless pose extremely interesting and demanding questions that do intersect with our core interests.

2.2 ACT vis-a-vis LAS

The ACT system embodies an extremely powerful thesis. That is, one set of learning principles underlies the whole gamut of human learning--from a child's first language to a graduate student's study of a text in abstract algebra. The four topics listed above, second

language learning, mathematics, programming, text study skills, are a fairly diverse sample of learning phenomena. If we can establish the adequacy of ACT for these domains we think we will have provided this bold thesis with some credibility. We do not see any a priori reasons to be predisposed to accept or reject this thesis. However, the consequences of this thesis, if true, are so enormous and important that it demands that someone explore it. Of course, this is just the thesis that underlies the older learning theories. However, it deserves another look in the framework of the current theoretical sophistication of cognitive science.

This thesis is diametrically opposed to that advocated by Chomsky (1965) and others to the effect that there are special mechanisms for language learning. This is a thesis that underlies some of my earlier work on the LAS model of language acquisition (Anderson, 1974, 1975, 1977, 1978). LAS had some special mechanisms to facilitate the acquisition of natural languages. We now think that these special mechanisms were really manifestations of more general learning abilities.

There were a number of inadequacies in the LAS program that underlies that earlier work. (These inadequacies are reviewed in Anderson, 1978.) These included inabilities to make discriminations, to correct errors, to deal with non-hierarchical aspects of language, to deal with inflections, to properly handle the non-declarative aspects of language, to properly model human limitations in language learning and performance, and to account for the gradualness of human learning. One

way or another each of these problems could have been handled by additions to the LAS theory--but at great cost to the overall parsimony and elegance of that theory. It seemed that a more elegant resolution was possible only by stepping back to a more general learning approach. We expect that ACT will reproduce many of LAS's learning feats and in ways similar to LAS; however, it will do so in a way that will naturally extend to the many problems LAS could not handle. Thus, LAS established what could be done by a set of learning mechanisms. ACT is an attempt to generalize upon what we have learned from LAS.

2.3 Overview of Paper

The remaining portion of the paper has the following organization. In Section 3, we will discuss the role computer simulation plays in the development of the ACT theory. In Section 4, we will describe that portion of the ACT system which is concerned with performing learned skills. This will be a brief update on the ACT system as described in Anderson (1976). There will not be attempts to document these features empirically as this has been done elsewhere (Anderson, 1976; Anderson & Kline, 1977). This part of the system closely corresponds to ACTE which served as the basis for the book, Language, Memory, and Thought. The current system, ACTF, differs from ACTE principly in its facilities for dealing with acquisition of productions. The remaining sections, but one, will be concerned with reviewing the principal new developments in the ACT theory that are concerned with

learning. Section 5 will discuss how new productions are entered into the system; Section 6, will discuss how productions are strengthened; Section 7, how productions become more general in their range of application; Section 8, how productions can be made more discriminative; and Section 9, how multiple productions get combined into a single production. Then Section 10, the last section, will contain evaluations, conclusions, and goals for the future. In particular this section will announce some of the new architectural developments we have in mind for the next ACT system--ACTG.

2.4 Status of this Paper

This paper is intended to serve three functions. First, it fulfills the yearly report requirement of the Office of Naval Research. Second, it will be used to inform interested colleagues of our work. Third, it serves as a preliminary draft for a number of published reports. Parts of this paper will appear in Aptitude, Learning, and Instruction: Cognitive Processes Analyses edited by Snow, Federico and Montague and in the 1979 volume 13 of Bower's Learning and Motivation. Because the paper is serving as a draft, we will describe work that is not yet completed but that we anticipate completing in time for the final drafts. In particular, some of the mini-simulations have not been implemented or are only partially implemented. We intend to complete these implementations before the due dates for the published reports. We identify any aspect of the paper which is describing incomplete or projected developments.

3 ACT as a Computer Simulation Program

3.1 A Programming Language and a Theory of Cognition

In this section, we will be partially building on points developed in Anderson & Kline (1977). The ACT system is a somewhat novel entity in psychology and it is necessary to carefully explain its rationale.

ACT is at the same time a high-level programming language and a theory of the mechanisms of human cognition. A high-level programming language is a formalism that facilitates programming certain kinds of algorithms. However, a high level language can make it difficult to program algorithms other than the class for which it was intended. For instance, it is difficult to do matrix multiplication in LISP. It is for this reason that *high-level programming* languages are often "special-purpose." ACT is a special-purpose programming language in this sense. The fact that certain procedures can be coded in ACT more efficiently and easily than other procedures is the means by which ACT provides a theory of cognitive mechanisms. Humans are also more successful at certain cognitive processes than others. The hope is that ACT limitations correspond to human limitations. For instance, humans find it harder to perform mental multiplication than mental addition and ACT does the same--in ACT's case because of the difficulty of keeping active the intermediate products needed in mental multiplication.

There is a classic distinction between learning and performance.

The ACT theory as developed in the 1976 book was an attempt to specify the performance limitations on previously acquired behaviors. The theory was concerned with characterizing the speed of basic operations and the probability of error in these operations. These limitations were built into the programming language. In developing the learning theory we are again concerned with the limitations--but this time the limitations on the acquisition process.

It might seem wrong-headed to focus on the limitations of human behavior, given that what is so remarkable about human behavior is its intelligence. However, it is our argument that it is these constraints that make intelligence possible. These constraints cut down on the systems options and force it to develop in certain directions rather than others. Thus, these constraints simplify the evolution of intelligent behavior. The highest level claim of the ACT theory is that the constraints which ACT, as a programming language, imposes on programming correspond to the constraints human cognitive mechanisms impose on the evolution of cognitive procedures.

3.2 Theories vs. Models

A distinction that one sometimes encounters is that between a theory and a model. This distinction has a clear analogue within the ACT programming framework. The ACT theory corresponds to the general rules for interpreting an ACT program and applying it to an ACT data base. Each program that can be created in the ACT language constitutes another

model for how a task can be performed. It is possible to create multiple programs within ACT for doing a particular task, say mental addition. Each of these programs constitutes a different model for how the task might be done. The ACT theory provides general principles about how such programs execute, where they are likely to break down, how long they take to apply, etc. It is a claim of ACT, as a theory, that any program that can be written in the language constitutes a valid psychological model for the task. The fact that there are many ways to program ACT to do a task corresponds to the fact that there are many ways people can and do perform tasks like mental addition. To disconfirm a particular ACT model of a particular subject doing a task it would be necessary to show that the model mispredicted the behavior of the subject. To show that ACT was wrong as a theory for a particular task it would be necessary to show that no model, programmable within ACT, predicted the behavior of some subject or that there was some model programmable in ACT which we could not get a subject to emulate.

These considerations create a certain buffer between ACT and its potential disconfirmation by data. What we usually propose and test are specific ACT models that seem plausible for a task. To disconfirm the model does not disconfirm the theory. It is always possible that some other model would account for the data. However, ACT as a theory would be rather useless if we did not have a way of deriving from the theory the models that are likely to account for behavior in a particular task. In fact, we do have a way of arriving at likely models for a task. This

is to use our knowledge as programmers as to what will be optimal, given the constraints of the ACT programming language. This is the principle of optimality suggested by Newell (1973). The basic assumption is that the human system tends to program itself in optimal ways. The consequence of these considerations is a means for discrediting the ACT theory: It is enough to show that no optimal ACT program can account for the data in a task. It is not necessary to consider the myriads of non-optimal programs (models).

3.3 Comparisons to Older Learning Theories

In some ways, ACT is really not that foreign to psychology. It is an attempt to provide a systematic theory of the potential of an organism like a classical learning theory such as that of Clark Hull. It is true that the assumptions of the theory are somewhat different, showing the influence of twenty-five years of advance in computer science, linguistics, and cognitive psychology. It shows a concern for what has been called "sufficiency conditions." That is, the concern is that the theory be capable of displaying behavior as complex and intelligent as we now know human behavior to be. We, however, do not think the assertions of the ACT theory are that different from what Hull might propose were he alive today and willing to try to take into account the past twenty-five years of advance in knowledge.

The big difference between ACT and past theories is its computer implementation--its realization as a programming language. It would be

possible to claim that this difference is purely technical, that the computer implementation provides a way of formalizing the theory, establishing its internal consistency, and testing its predictions. This was not properly done for the Hullian theory, which evolved before the computer, and certainly could not be done for a theory like ACT without a computer implementation.

3.4 Optimality and Efficiency

While the computer implementation does serve this critical, technical function for the ACT theory, it is also the case that our use of the computer implementation has subjected our theory construction to constraints of a type not previously used in psychology. These are the constraints of optimality and efficiency of performance. These two considerations are related. Efficiency refers to the constraint that one should not propose mechanisms which are operating in obviously inefficient manners. Optimality refers to the constraint that one does not propose mechanisms that will cause the system to behave in a non-adaptive way. Thus efficiency is more concerned with performance while optimality is more concerned with learning.

The constraints of efficiency and optimality are quite potent. Consider the constraint of efficiency as applied to the serial, exhaustive scanning explanation of the Sternberg paradigm (1969) that does a good job of accounting for the data. That model proposes that a subject, in searching for a target item in a short-term memory set, will

serially compare the target against each memory set item. These comparisons continue until the set is exhausted, even past the point where the target item has matched a memory set item. It seems inefficient to propose that short-term memory wastes times making comparisons after the point that the desired match has been obtained. The assumption of a serial exhaustive search has been justified by the fact that subjects' decision time increases at the same linear rate with memory set size for positive and negative targets. In defending the exhaustive assumption Sternberg has argued that it may be more expensive to check a signal to see if a match occurred than to make another match of characters. While no doubt such a system could be built that found it more expensive to check a signal than to make a match it does seem a rather perverse design. As Huessman & Woocher (1976) have argued, it should be much easier to detect a 0-1 signal than to detect the presence of complex characters. The criterion of efficiency would weigh heavily against an exhaustive, serial match for the memory scanning. However, there are a number of parallel processing models which could predict the same linear functions for positive and negative items without violating the criterion of efficiency (Anderson, 1976; Townsend, 1971, 1974). Thus, the principle of efficiency points to a parallel model for the Sternberg paradigm.

For an illustration of the potency of the principle of optimality, consider the literature on child language acquisition. There have been frequent observations about the relative ineffectiveness of

negative information (e.g., Braine, 1963; Brown, 1973; Cazden, 1965). It has been observed that children derive little benefit from information about what are incorrect syntactic structures. Consider, for instance, a child who has overgeneralized the pluralization rule and is generating "foots." Since some terms do permit variable pluralization (e.g., two fish or two fishes) how is the child to know he is wrong unless he gets negative feedback and can take advantage of it? Therefore, the principle of optimality would deny the conclusion that children do not use negative information. We will discuss later (page 97) how and when negative information can be effective in the ACT system.

It has been argued (Anderson, 1976) that psychological theory is seriously underdetermined by behavioral data--that there are many ways to account for such phenomena as the equal effects of memory set on positive or negative probes or the apparent ineffectiveness of correction on child language development. The criteria of efficiency and optimality offer means to further constrain theorizing to help produce a more unique psychological explanation of the phenomena.

3.5 Good Cognitive Psychology is Good Artificial Intelligence

While it is possible to discuss the criteria of efficiency and optimality in the abstract, to be able to rigorously and thoroughly apply these criteria we need a model of the physical system to which they apply. Unfortunately, we know hardly any relevant information about the human nervous system. In the ACT project we have proposed what might

seem like a preposterous suggestion: to let our knowledge of modern computer technology serve as a surrogate for knowledge of human hardware. Specifically, we propose to relate psychological efficiency and optimality constraints to what is efficient and optimal for our computer implementation. The claim (Anderson & Kline, 1977) is that an accurate psychological model would not become unacceptably inefficient or non-optimal when implemented on a modern computer. This claim is part of a larger conjecture about the relationship between artificial intelligence and cognitive psychology which was stated as:

> (1) Good cognitive psychology is good artificial intelligence

by which it was meant that good cognitive psychology mechanisms can be simulated in a way that would constitute good artificial intelligence programs. This means that a side benefit of cognitive psychology could be the development of good artificial intelligence theory.

Our conception of the relationship between artificial intelligence and cognitive psychology is illustrated by the Venn diagram in Part a of Figure 1. There is the space of all possible mechanisms; a subset of these are good artificial intelligence mechanisms; and a subset of these are good models for cognitive psychology. This is to be contrasted with Part b of Figure 1 which represents an alternative that might seem, at least a priori, more plausible. In this figure, good AI mechanisms and good psychology mechanisms only overlap. There are good psychological mechanisms which do not constitute good artificial intelligence.

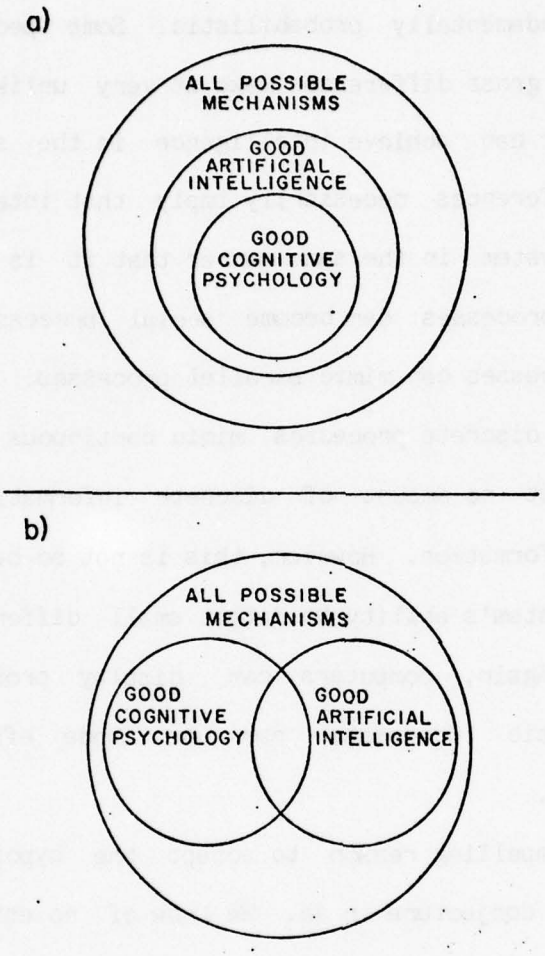


Figure 1. Venn Diagrams illustrating possible relationships between cognitive psychology and artificial intelligence.

There are a number of reasons why Part b of Figure 1 might seem more plausible than Part a. There are a number of features of the human

brain which do not correspond to the serial digital computer. Among these are the fact that it processes information in a highly parallel manner, it processes in terms of continuously varying quantities, and its behavior appears to be fundamentally probabilistic. Some people feel that the existence of these gross differences make it very unlikely that the brain and the computer can achieve intelligence in the same way. However, none of these differences necessarily imply that intelligence cannot be achieved in one system in the same manner that it is achieved in the other. Parallel processes can become serial processes as a special case and serial processes can mimic parallel processes. Again it is not difficult to have discrete processes mimic continuous or vice versa. It might seem that no amount of discrete information could exactly mimic continuous information. However, this is not so because of limitations of any real system's ability to detect small differences in continuous quantities. Again, computers can display probablistic behavior and probabilistic processes can be made effectively deterministic by redundancy.

We know of no compelling reason to accept the hypothesis in Figure 1b over the stronger conjecture in 1a. We know of no established psychological or physiological mechanisms that cannot be modelled with satisfactory efficiency in a computer. (Admittedly, this is largely a statement about current ignorance as to psychological and physiological mechanisms.) Therefore, we have chosen to subscribe to the conjecture represented in Part a of Figure 1 as a useful heuristic in constraining

our theorizing. The implication is that implementation considerations should play a critical role in decisions about the future development of the ACT theory.

There is another motivation for this heuristic, in addition to enabling a rigorous and thorough application of efficiency and optimality constraints. Recall that the computer simulation is the only technical means currently available of formalizing the ACT theory, of establishing its completeness, and of providing a rigorous basis for deriving predictions. If we allowed it to perform in grossly inefficient or non-optimal ways, the simulation would become intractable and fail to fulfill its technical purpose. Thus, another justification for enabling implementation to determine theory is a standard one in science--we cannot let our theory go in directions that would make it unmanageable. In past generations this constraint was manifested by pleas for "mathematical tractability." Today, the plea is for "computational tractability." The computer has enabled us to think about things we could not think about before. We submit that the contrast between ACT and Hullian theory is one example of that. However, even today there are technical limits on what we can think about.

4 ACT as a Performance Model

A central design feature of the current ACT system is the distinction between procedural knowledge and declarative knowledge. Procedural knowledge is knowledge about how to do various things like

solving mathematical problems, writing a computer program, or understanding a spoken sentence. Declarative knowledge provides the data for various cognitive procedures. It includes knowledge about the general world such as: "George Washington was the first president of the United States." The declarative knowledge is represented in a propositional network data base while the procedural knowledge is described as a set of productions. We will describe ACT's propositional network and then its production system.

4.1 ACT's Data Base

Figure 2 illustrates a fragment of ACT's longterm memory network. It encodes the fact that Mommy gave a candy to Daddy and illustrates some of the structure connected to the concepts involved in that fact. Anderson (1976, Ch. 5) gives a complete specification of the features contained in this network representation. We will just highlight some of the important features. Knowledge is represented as a large network of nodes, which represent ideas, that are interconnected by labelled links, which represent various types of associations between the ideas. Information is organized into propositional units where each proposition is a tree interassociating a number of nodes. There is a distinction made between words and the concepts they reference. There are links labelled W interconnecting words and their concepts. The standard convention is to represent a concept for a word by that word prefixed by "@." There is a distinction made between nodes representing a general class like @CANDY and nodes representing specific instances like CANDY1.

Nodes representing a specific instance, such as CANDY1, are referred to as tokens. Every input to the data base is a distinct event and, therefore, is represented by a new token.

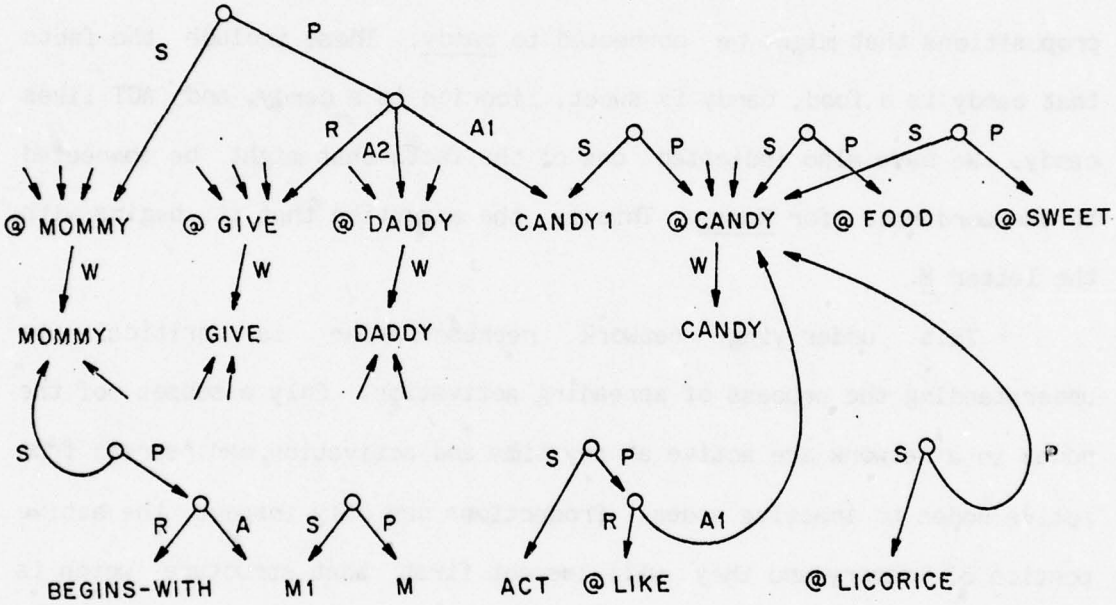


Figure 2. A fragment of ACT's propositional network surrounding the concept @CANDY.

While these nodes have been given mnemonic labels to help communicate their meaning to the reader, their labels are irrelevant to their treatment in the theory or in the simulation program. The meaning of a node is given by the network structure in which it is embedded and the procedures which operate on that network structure. Each of the word nodes and concept nodes involved in Mommy gave a candy to Daddy is involved in a large number of other associations which were in memory before the encoding of this assertion. We have shown some of the propositions that might be connected to candy. These include the facts that candy is a food, candy is sweet, licorice is a candy, and ACT likes candy. We have also indicated one of the facts that might be connected to the word node for Mommy. This is the assertion that it begins with the letter M.

This underlying network representation is critical to understanding the process of spreading activation. Only a subset of the nodes in a network are active at any time and activation can spread from active nodes to inactive nodes. Productions can only inspect the active portion of memory and they will inspect first that structure which is most active. Thus activation serves a working memory role and serves to enable the system to focus its computational resources.

Spreading activation will not be critical to the learning discussion in this paper. While helpful to an understanding of spreading activation, for other purposes, the ACT network notation can be quite cumbersome. Therefore, we will use a linearized representation of the

propositional structure which is easier to work with. These linearizations will represent each assertion in the network as a list consisting of, first, the relation or predicate and, then, a list of its arguments. A linearized representation of Figure 2 is given in Table 1. Note the special structure set up to represent the links between words and their concepts. We will use such linearizations throughout the remainder of the paper, but the reader should bear in mind that what is really assumed, both in theory and implementation, is a network representation like Figure 2.

Table 1
Linearization of the information in Figure 2

```
@GIVE @MOMMY @DADDY CANDY1)
(@CANDY CANDY1)
(WORD-FOR @MOMMY MOMMY)
(WORD-FOR @GIVE GIVE)
(WORD-FOR @DADDY DADDY)
(WORD-FOR @CANDY CANDY)
(@FOOD @CANDY)
(@SWEET @CANDY)
(@CANDY @LICORICE)
(@LIKE ACT @CANDY)
(BEGINS-WITH MOMMY M1)
(M M1)
```

4.2 Productions in ACT

ACT's procedural knowledge is encoded as a set of productions. The ACT production system can be seen as a considerable extension and modification of the production systems developed at Carnegie-Mellon (Newell, 1972, 1973; Rychener & Newell, 1977). Each production is a condition-action pair. The condition is a pattern describing a piece of

the data base (declarative component). If the condition of a production matches the data base, the action of the production will be executed. Actions can both add to the contents of the data base and cause the system to emit responses. Each production is like an independent demon that is inspecting the data base to see if its condition is satisfied. It is possible for more than one production to apply at a time, giving the system an important parallel processing capacity.

In the condition of a production the description of network structure can be specific, naming the particular nodes and connections which are required, or variables may be used to designate nodes whose exact identity is unimportant or unknown in advance. A variable can stand for any node in memory, as long as it is connected to the specific nodes named in the condition in the way laid out in the description. To make this work, every description must contain at least one specific node--it cannot have only variables. Consider the following condition:

```
(@EAT @MOMMY LVar1)
```

It specifies that there be a structure in the data base of the form "Mommy eat something." The variable LVar1 in the condition stands for the something. All elements beginning with the initials LV are local variables. Local variables can match anything when they occur in a condition. This structure could match any number of structures in the data base such as:

```
(@EAT @MOMMY @SOUP)
(@EAT @MOMMY @YOGURT)
```

Consider the following complete production:

```
(EAT @MOMMY LVar1)
```

```
ABS (@EDIBLE LVar1)
=>
(@EDIBLE LVar1)
(GVar = LVar1)
```

The convention for representing productions is to give the condition above the arrow and the action below. This production illustrates a number of additional features of ACT productions. The condition includes an absence test--indicated by the ABS element. This specifies the constraint that the element which matches LVar1 must not have the predicate @EDIBLE stored with it. Thus, this production will apply if there is something in the data base marked as being eaten by Mommy but which is not marked as edible. If there is such a thing, the action of the production marks it as being in the class @EDIBLE. Thus this production makes the inference that something Mommy eats is edible. The action also illustrates one operation that might be performed on global variables. A global variable is an element beginning with the initials GV. In this case, the global variable GVar is set to the value of the local variable LVar1. Global variables differ from local variables in that they keep their values beyond a single production and can be used to communicate among productions. Global variables thus serve as a special sort of short-term memory. There is a limitation on the number that can be simultaneously bound. Currently that limit is 10. In contrast, local variables only retain their values within a single production. Local variables can be bound in matching the condition and serve to communicate to an absence test or to the action.

4.3 Example Production System[2]

It would be useful to present a small set of productions for performing a particular task. The set of productions we have chosen is concerned with performing addition of two numbers. Table 2 contains this set of productions. That production set assumes that the addition problem is posed as an instruction to add two numbers--e.g., (ADD NUMB1 NUMB2)--where the two numbers are represented by arbitrarily denoted nodes. Additional propositions encode the sequence of digits defining the to-be-added numbers. So the problem 832 + 418 would be represented:

```
(ADD NUMB1 NUMB2)

(BEGINS NUMB1 TOK11)
(2 TOK11)
(AFTER TOK11 TOK12)
(3 TOK12)
(AFTER TOK12 TOK13)
(8 TOK13)
(ENDS NUMB1 TOK13)

(BEGINS NUMB2 TOK21)
(8 TOK21)
(AFTER TOK21 TOK22)
(1 TOK22)
(AFTER TOK22 TOK23)
(4 TOK23)
(ENDS NUMB2 TOK23)
```

Consider the encoding of the first number, NUMB1 (832): The number is encoded from right to left. It is stated that NUMB1 begins on the right with a token TOK11. TOK11 is a token of the digit 2. TOK12 is after TOK11 and TOK12 is a token of the digit 3. TOK13 is after TOK12 and TOK13 is a token of the digit 8. NUMB1 ends on the left with TOK13.

Figure 3 illustrates the flow of control among the productions in

Table 2. The productions are represented as arrows connecting states represented by circles. Each arrow is labelled by the production which it represents. The state circle at the head of an arrow shows the action of the production. The arrows for other productions which need these actions performed in order to apply are shown originating from this state circle. Other, static information from the data base must be examined in order to decide which production should apply when two or more productions originate from a state circle. Such additional conditions are represented in diamonds adjacent to production numbers. The state circle at the tail of a production arrow along with the adjacent diamond represent the condition of that production.

TABLE 2
Productions to Perform Addition of Two Numbers

```
P1:     (LVgoal = (ADD LVfirst LVsecond))
        (BEGINS LVfirst LVtok1)
        (BEGINS LVsecond LVtok2)
        =>
        (ADD LVtok1 LVtok2)
        (GVgoal = LVgoal)
        (GVtok1 = LVtok1)
        (GVtok2 = LVtok2)

P2:     (ADD GVtok1 GVtok2)
        (LVdigit1 GVtok1)
        (LVdigit2 GVdigit2)
        (PLUS LVdigit1 LVdigit2 LVsum)
        =>
        (PUT-OUT LVtok3)
        (LVsum LVtok3)

P3:     (ADD GVtok1 GVtok2)
        (LVdigit1 GVtok1)
        (LVdigit2 GVtok2)
        (PLUS LVdigit1 LVdigit2 LVsum)
        (CARRY GVtok1)
        =>
        (PUT-OUT LVtok3 CARRIED)
        (LVsum LVtok3)

P4:     (PUT-OUT LVtok3)
        (LVsum LVtok3)
        (PUT-OUT LVtok3 CARRIED)
        ABS (GREATER LVsum 9)
        =>
        (WRITE LVsum)
        (DO-NEXT GVtok1 GVtok2 NOCARRY)

P5:     (PUT-OUT LVtok3 CARRIED)
        (LVsum LVtok3)
        (PLUS 1 LVsum LVsum1)
        ABS (GREATER LVsum1 9)
        =>
        (WRITE LVsum1)
        (DO-NEXT GVtok1 GVtok2 NOCARRY)
```

```
P6:     (PUT-OUT LVtok3)
        (LVsum LVtok3)
        (GREATER LVsum 9)
        (PLUS LVdigit3 10 LVsum)
        ABS (PUT-OUT LVtok3 CARRIED)
        =>
        (WRITE LVdigit3)
        (DO-NEXT GVtok1 GVtok2 CARRY)

P7:     (PUT-OUT LVtok3 CARRIED)
        (LVsum LVtok3)
        (GREATER LVsum 8)
        (PLUS LVdigit3 9 LVsum)
        =>
        (WRITE LVdigit3)
        (DO-NEXT GVtok1 GVtok2 CARRY)

P8:     (DO-NEXT GVtok1 GVtok2 NOCARRY)
        (AFTER GVtok1 LVtok3)
        (AFTER GVtok2 LVtok4)
        =>
        (ADD LVtok3 LVtok4)
        (GVtok1 = LVtok3)
        (GVtok2 = LVtok4)

P9:     (DO-NEXT GVtok1 GVtok2 NOCARRY)
        (AFTER GVtok1 LVtok3)
        (LVdigit3 LVtok3)
        (ENDS LVsecond GVtok2)
        =>
        (WRITE LVdigit3)
        (DO-NEXT LVtok3 GVtok2 NOCARRY)
        (GVtok1 = LVtok3)

P10:    (DO-NEXT GVtok1 GVtok2 NOCARRY)
        (AFTER GVtok2 LVtok4)
        (LVdigit4 LVtok4)
        (ENDS LVfirst GVtok1)
        =>
        (WRITE LVdigit4)
        (DO-NEXT GVtok1 LVtok4 NOCARRY)
        (GVtok2 = LVtok4)
```

```
P11:    (GVgoal = (ADD LVfirst LVsecond))
        (DO-NEXT GVtok1 GVtok2 NOCARRY)
        (ENDS LVfirst GVtok2)
        (ENDS LVsecond GVtok2)
        =>
        (DONE GVgoal)
        (UNBIND GVgoal GVtok1 GVtok2)

P12:    (DO-NEXT GVtok1 GVtok2 CARRY)
        (AFTER GVtok1 LVtok3)
        (AFTER GVtok2 LVtok4)
        =>
        (ADD LVtok3 LVtok4)
        (CARRY LVtok3)
        (GVtok1 = LVtok3)
        (GVtok2 = LVtok4)

P13:    (DO-NEXT GVtok1 GVtok2 CARRY)
        (AFTER GVtok1 LVtok3)
        (LVdigit3 LVtok3)
        (PLUS 1 LVdigit3 LVsum)
        (ENDS LVsecond LVtok2)
        ABS (EQUAL 10 LVsum)
        =>
        (WRITE LVsum)
        (DO-NEXT LVtok3 GVtok2 NOCARRY)
        (GVtok1 = LVtok3)

P14:    (DO-NEXT GVtok1 GVtok2 CARRY)
        (AFTER GVtok2 LVtok4)
        (LVdigit4 LVtok4)
        (PLUS 1 Vdigit4 LVsum)
        (ENDS LVfirst GVtok1)
        ABS (EQUAL 10 LVsum)
        =>
        (WRITE LVsum)
        (DO-NEXT GVtok1 LVtok4 NOCARRY)
        (GVtok2 = LVtok4)
```

```
P15:    (DO-NEXT GVtok1 GVtok2 CARRY)
        (AFTER GVtok1 LVtok3)
        (LVdigit3 LVtok3)
        (9 LVtok3)
        (ENDS LVsecond GVtok2)
        =>
        (WRITE 0)
        (DO-NEXT LVtok3 GVtok2 CARRY)
        (GVtok1 = LVtok3)

P16:    (DO-NEXT GVtok1 GVtok2 CARRY)
        (AFTER LVtok2 LVtok4)
        (LVdigit4 LVtok4)
        (9 LVtok4)
        (ENDS LVfirst GVtok1)
        =>
        (WRITE 0)
        (DO-NEXT GVtok1 LVtok4 CARRY)
        (GVtok2 = LVtok4)

P17:    (GVgoal = (ADD LVfirst LVsecond))
        (DO-NEXT GVtok1 GVtok2 CARRY)
        (ENDS LVfirst GVtok1)
        (ENDS LVsecond GVtok2)
        =>
        (WRITE 1)
        (DONE GVgoal)
        (UNBIND GVgoal GVtok1 GVtok2)
```

Production P1 is responsible for starting the processing of the information. The first element in the condition of P1 is (LVgoal = (ADD LVfirst LVsecond)). This matches the first element in the problem description and sets LVgoal to the structure. We will indicate this assignment of memory structure to pattern as follows:

(LVgoal = (ADD LVfirst LVsecond)) = (ADD NUMB1 NUMB2)

The other two segments in the condition of P1 match elements in the problem description:

(BEGINS LVfirst LVtok1) = (BEGINS NUMB1 TOK11)
(BEGINS LVsecond LVtok2) = (BEGINS NUMB2 TOK21)

P1 adds to memory a self-instruction to add the first two digits: (ADD TOK11 TOK12). It also sets the global variable, GVgoal, to LVgoal which is bound to the problem. This serves to focus ACT's attention on the problem at hand. Finally, GVtok1 is set to LVtok1 and GVtok2 is set to LVtok2. This maintains ACT's attention on the tokens of the individual digits which it is about to add.

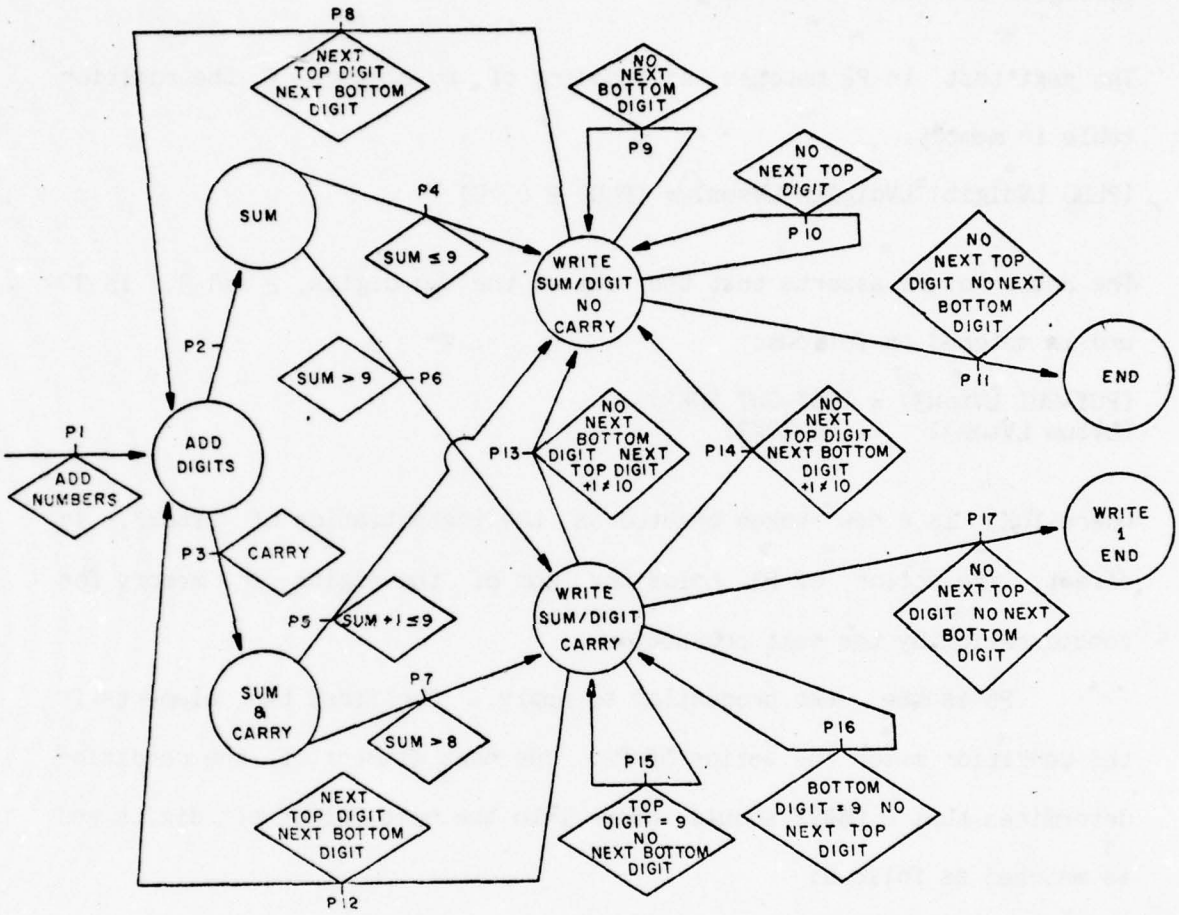


Figure 3. The flow of control among the productions in Table 2.

After the execution of production P1 and the introduction of (ADD TOK11 TOK21) into the data base, production P2 is the only production whose condition matches. The first three elements in the condition of P2 match elements of the problem encoding as indicated below:

(ADD GVtok1 GVtok2) = (ADD TOK11 TOK21)
(LVdigit1 GVtok1) = (2 TOK11)
(LVdigit2 GVtok2) = (8 TOK21)

The next test in P2 matches an encoding of an element of the addition table in memory:

(PLUS LVdigit1 LVdigit2 LVsum) = (PLUS 2 8 10)

The action of P2 asserts that the sum of the two digits, 2 and 8, is 10 and is matched as follows:

(PUT-OUT LVtok3) = (PUT-OUT TOK3)
(LVsum LVtok3) = (10 TOK3)

where TOK3 is a new token created as the instantiation of LVtok3. In effect, the action of P2 holds the sum of the digits in memory for consideration by the next production.

P6 is the next production to apply. The first two elements in the condition match the action of P2. The next element of the condition determines that a CARRY should occur into the next column of digits and is matched as follows:

(GREATER LVsum 9) = (GREATER 10 9)

The data base contains the fact that 10 is greater than 9: (GREATER 10 9). The condition then determines the digit to be written out as part of the solution through the match:

(PLUS LVdigit3 10 LVsum) = (PLUS 0 10 10)

The absence condition, ABS (PUT-OUT LVtok3 CARRIED), insures that P6 will

not misapply when P7 should apply. P6 will apply only when CARRIED has not been indicated.

The action of P6 writes the digit 0 and sets up an advance of attention to the next column of digits with the information that a CARRY occurs into that next column. With replacement of variables by their values these actions become:

```
(WRITE LVdigit3)                = (WRITE 0)
(DO-NEXT GVtok1 GVtok2 CARRY) =  (DO-NEXT 2 8 CARRY)
```

It is worthwhile considering why only P6 applies and not some of the other productions. P4 does not apply because of the absence test, ABS (GREATER LVsum 9), 10 is greater than 9. P5 and P7 do not apply because CARRIED does not occur in the proposition (PUT-OUT LVtok3). One might wonder why P1 does not apply again. Its condition was satisfied once and nothing has happened to change the data elements which satisfied it. This illustrates an important principle of production interpretation: It is not possible to match a production condition twice to exactly the same data base elements.

The next production to apply is P12. It matches further elements of the problem description and retrieves the tokens of the next column of digits to be added. Attention is maintained on these digits by assigning them to global variables. That a CARRY should be added into this column is again flagged. P12 adds the following structures to the data base:

```
(ADD TOK12 TOK22)
(CARRY TOK12)
```

With these elements added into the data base production P3 will apply adding the second column digits (3 and 1) plus the CARRY to achieve a sum of 5. This production adds to the data base the elements:

(PUT-OUT TOK3)
(5 TOK3)

Production P5 then applies writing out the digit 5 and setting up an advance to the third column of digits without a CARRY. Production P8 accomplishes this shift of attention to the third column.

Production P2 next applies adding 8 and 4 and holding the sum of 12 for consideration. Production P6 writes the digit 2 and flags a CARRY. Production P17 completes the problem by writing a 1 into a new column in the problem solution.

This example illustrates a number of important features of production system execution:

(1) Individual productions act on the information in long-term memory. They communicate with one another by entering information into memory.

(2) Productions tend to apply in sequences where one production applies after another has entered some element into the data base. Thus the action of one production can help evoke other productions.

(3) Productions respond to patterns of events. The more clauses specified in the condition of the production the more restricted the set of patterns that will evoke it. The more local variables and the fewer nodes, the more unrestricted the set of evoking patterns.

This concludes our discussion of the basic properties of the ACT performance system. A fuller description of the significant properties of this system can be found in Anderson (1976), Anderson, Kline, & Lewis (1977), and Anderson & Kline (1977). Before turning to a discussion of learning, we would like to make some remarks about the implementation status of this system: The original system described in Anderson (1976) is called ACTE, and is being maintained as a courtesy system for other users. It is described in Kline & Anderson (1976). The system will be superceded by an upward compatible system called ACTF which contains most of the learning properties described in the subsequent sections. It also contains some improvements to the performance system. A user's manual for this system is in preparation. There is yet another system, ACTG, in the planning stages. This is projected to contain some of the improvements proposed at the end of the paper.

5 Production Designation

ACT needs to have the capacity to add new productions. Productions can designate the construction of other productions in their actions just as they can designate the construction of memory structure. Production designation is an important means by which ACT learns procedural skills. However, just as the building of memory structure serves functions other than declarative learning, so it is the case that production designation serves functions other than procedural learning. Creating new memory structure may serve the function of communicating to

other productions. Similarly, creating new productions can serve as a means of getting a specific behavior accomplished. Because a production is so much more complex than a proposition, the factors governing production designation are correspondingly more complex than the factors governing designation of memory structure. This section will discuss and illustrate both the complexities of production designation and the function of designation.

5.1 The Notation of Designation

To introduce the standard notation for designation, we will consider a simple example of how the capacity for designation can be used to comply with requests of the form: Point to the letter (e.g., Point to the H.) This statement is a request to retrieve the pattern definition of the letter from the data base, match it to the presented letters, and point to the appropriate item. Suppose, the definition of H is stored in the data base as:

```
S1:     (PATTERN H ((CONSISTS OBJ L1 L2 L3)
                    (VERTICAL L1)
                    (VERTICAL L2)
                    (HORIZONTAL L3)
                    (BISECT L3 L1)
                    (BISECT L3 L2)))
```

This encodes a list of features that an object OBJ must possess to be an instance of the letter H. The following production responds to the appearance of a pointing request, retrieves such a pattern description, and designates a production which will perform the desired pointing:

```
P1:     (POINT-TO-THE-LVletter)
        (PATTERN LVletter LVpat)
        (LVpat = ((CONSISTS LVobj...)...))
        =>
        (CONDITION (ON LVobj SCREEN) LVpat)
        (ACTION (POINT ACT LVobj))
```

The first structure in the condition responds to the appearance of the request. The second retrieves the desired pattern and the third sufficiently unpacks the pattern so as to bind the object described to LVobj. To an H request this would match with the following binding of variables: LVletter = H, LVpat = H description, LVobj = OBJ. The action of this production designates a new production by specifying condition and action. The condition of this designated production asserts that LVobj be on the screen and also includes all the description of LVobj in the pattern LVpat. The action is for ACT to point to LVobj. With these bindings for the variables in the designating production P1 the following designated production would be produced:

```
P2:     (ON LVobj1 SCREEN)
        (CONSISTS LVobj1 LVline1 LVline2 LVline3)
        (VERTICAL LVline1)
        (VERTICAL LVline2)
        (HORIZONTAL LVline3)
        (BISECT LVline3 LVline1)
        (BISECT LVline3 LVline2)
        =>
        (POINT ACT LVobj1)
```

If there were an object on the screen matching this description ACT would point to it. Note that the nodes OBJ, L1, L2, L3 in the structure S1 are replaced by variables LVobj1, LVline1, LVline2, and LVline3 in the designated production. This is because these nodes are flagged in memory

as indefinite. When an indefinite node is used in designating a production it is replaced in the designated production by a new variable.

This example illustrates the basic technique of production designation. The designating production indicates the structures that are to go into the condition and action. It does this either by pointing to existing structures in memory--as with the definition of H or it does it by placing in CONDITION or ACTION the appropriate structure, as with (ON LVobj SCREEN) in CONDITION and (POINT ACT LVobj) in ACTION.

This example, however, does not really involve any learning. The production created will not be used again. It was simply created to cause a particular pattern to be matched in one situation and have a behavior generated. That is, production designation allows ACT to mobilize all of its pattern matching capabilities to accomplish a particular task. The remainder of Section 5 is concerned with how production designation can be used for learning.

5.2 Encoding of Procedural Instructions

As a first example of how production designation can be used in learning, let us consider how this device is used to create permanent productions to encode the lessons of direct instruction. As an example, consider how ACT might encode the following instructions defining various types of expressions in LISP (adapted from the first chapter of Weissman (1967)):

1. If an expression is a number it is an atom.

2. If an expression is a literal (a string of characters) it is an atom.

3. If an expression is an atom it is an S-expression.

4. If an expression is a dotted pair, it is an S-expression.

5. If an expression begins with a left parenthesis, followed by an S-expression, followed by a dot, followed by an S-expression, followed by a right parenthesis, it is a dotted pair.

Table 3 illustrates the four ACT productions required to process these sentences.[3] Production L1 handles the *if* phrase in sentences like (1) - (4). (In this production and the others in Table 3, we have omitted, for simplicity's sake, representing the distinction between words and their tokens which is critical to successful operations of the implemented system.) Consider how this production would apply to the first definition. The first clause of the condition matches the *if* phrase in the definition:

(IF-AN-EXPRESSION-IS-A-LVword) = (IF-AN-EXPRESSION-IS-A-NUMBER)

In making this match LVword is bound to NUMBER. The second clause of the condition matches a memory encoding of the connection between the word NUMBER and its corresponding idea:

(WORD-FOR LVword LVcategory) = (WORD-FOR NUMBER @NUMBER)

The variable LVcategory is bound to @NUMBER. The action creates a structure predicating @NUMBER of an object:

(LVcategory LVobj) = (NUMBER OBJ)

where a newly created node, OBJ, is assigned to the variable LVobj. (Unbound variables in the action of productions are given new nodes as values.) A global variable GVhold is set to this structure. This structure will be made the condition of a designated production. The global variable GVword is set to the last word processed in the if phrase and GVobj is set to OBJ.

Production L2 applies after L1. It matches the result phrase of the definition:

(GVword-IT-IS-A-LVword) = (NUMBER-IT-IS-A-ATOM)

The second clause in the condition of L2 matches the word-meaning connection:

(WORD-FOR LVword LVcategory) = (WORD-FOR ATOM @ATOM)

The action of L2 designates a production whose condition is patterned after the structure held by GVhold and whose action is patterned after the structure:

(LVcategory GVobj) = (@ATOM OBJ)

The actual production created is:

```
(NUMBER LVx)
=>
(ATOM LVx)
```

where LVx is a variable introduced to replace the indefinite node, OBJ, referred to by GVobj in the designating production.

Table 3
A set of Productions for Encoding Information
about LISP Structures

```
LI:     (IF-AN-EXPRESSION-IS-A-LVword)
(       (WORD-FOR LVword LVcategory)
        =>
        (GVhold (LVcategory LVobj))
        (GVword = LVword)
        (GVobj = LVobj)

L2:     (GVword-IT-IS-A-LVword)
        (WORD-FOR LVword LVcategory)
        =>
        (CONDITION GVhold)
        (ACTION (LVcategory GVobj))


L3:     (IF-AN-EXPRESSION-BEGINS-WITH-A-LVword)
        (WORD-FOR LVword LVcategory)
        =>
        (GVhold (BEGINS LVobj LVobj1)
                (LVcategory LVobj1))
        (GVword = LVword)
        (GVobj = LVobj)
        (GVobj1 = LVobj1)

L4:     (GVword-FOLLOWED-BY-A-LVword)
        (WORD-FOR LVword LVcategory)
        =>
        (GVhold (BEFORE GVobj1 LVobj1))
                (LVcategory LVobj1))
        (GVobj1 = LVobj1)
        (GVtok = LVtok)
```

Productions L3 and L4 are responsible for processing the more complex composite condition like (5). L3 processes the first begins phrase and L4 each subsequent followed-by phrase. They build up in propositional form a description of the structure described and attach

that description to GVhold. Thus, GVhold points to a set of descriptions. After the condition is complete, production L2 will apply to process the result clause and designate the production. The production designated after processing (5) is:

```
P2:     (BEGINS LVy LVa)
        (@LEFT-PARENTHESIS LVa)
        (BEFORE LVa LVb)
        (@S-EXPRESSION LVb)
        (BEFORE LVb LVc)
        (@DOT LVc)
        (BEFORE LVc LVd)
        (@S-EXPRESSION LVd)
        (BEFORE LVd LVe)
        (@RIGHT-PARENTHESIS LVe)
        =>
        (@DOTTED-PAIR LVy)
```

The production system in Table 3 represents a relatively pure instructional system. The output of these productions are other productions which serve the function of actually recognizing the various LISP expressions.

5.3 The Preprocessor

A set of designating productions like those in Table 3 are clearly a highly structured and sophisticated system. They represent the outcome of considerable learning about the nature of language and instruction. It turns out that it is not just in instructional situations that there exist sophisticated initial systems. In many learning situations properly called "inductive" there is an important contribution of sophisticated designating productions to learning. In all learning situations, instructional or inductive, we propose that

there exists a set of designating productions that serve to structure the learning situation. Depending on their sophistication they structure the situation to a greater or lesser extent. In all cases we refer to the set of relevant designating productions and productions which influence the designating productions as the preprocessor. Thus, the set of four productions in Table 3 constitute a preprocessor for encoding LISP definitions. The preprocessor does not contain all the intelligence contributing to learning but it does contain a good bit.

5.4 The Initial Preprocessor and First Language Acquisition

We are beginning to formulate a set of hypotheses about how children structure their initial learning situations and how this would apply to first language learning. Curiously enough, these proposals are not that different from classical learning theory proposals. Specifically, we are proposing a rather primitive set of productions for initially structuring learning. Table 4 contains what two of these productions might look like. Production INNATE1 encodes a principle of reinforcement. It asserts that if event LVevent occurred at time LVtime1, if ACT performs LVresponse at time LVtime2 and if ACT is reinforced at time LVtime3 and if these three events are in close temporal succession, then ACT will construct a production generating the required action in the prescribed situation.

Table 4
Two Initial Preprocessing Productions

```
INNATE1:(AT-TIME LVevent LVtime1)
        (LVresponse = (LVaction ACT LVobj))
        (AT-TIME LVresponse LVtime2)
        (LVresult = (REINFORCED ACT))
        (AT-TIME LVresult LVtime3)
        (JUST-BEFORE LVtime1 LVtime2)
        (JUST-BEFORE LVtime2 LVtime3)
        =>
        (CONDITION LVevent)
        (ACTION LVresponse)

INNATE2:(AT-TIME LVevent1 LVtime1)
        (AT-TIME LVevent2 LVtime2)
        (JUST BEFORE LVtime1 LVtime2)
        (INTERSECTING LVevent1 LVevent2)
        =>
        (CONDITION LVevent1)
        (ACTION (PREDICT ACT LVevent2))
```

To illustrate this production suppose the event was mother pointing at a ball, represented :

```
(POINT MOMMY X)
(@BALL X)
```

and by one means or another ACT was induced to say "ball," represented:

```
(SAY ACT BALL)
```

Then, if ACT was reinforced the production that would be designated would be:

```
(POINT MOMMY X)
(@BALL X)
=>
(SAY ACT BALL)
```

This sort of production is a modest but necessary step towards the lexicalization of natural language. That is, it introduces a connection between the word BALL and the concept @BALL.

In the ACT framework, the traditional contiguity constraints on the effectiveness of reinforcement become particularly meaningful. There are any number of events, actions, and reinforcements in one's life. If productions were being generated from all combinations there would be hopelessly many productions to deal with. The constraints of contiguity between event and response and between response and reinforcement serve to focus the system in on those combinations which are most likely to be relevant.

This focusing function is also seen strongly in production INNATE2 which builds into ACT a principle of association by contiguity and similarity. It will apply whenever two events, LVevent1 and LVevent2 occur in close temporal contiguity and whenever there is a network connection between them as tested by the INTERSECTING condition. The INTERSECTING condition successfully applies whenever the two events have a network intersection in memory (looking up to specified depth--currently 10 links). The intersecting test serves basically to weed out many of the accidental contiguities of events and serves to focus ACT on events which do have a relation.

As an example of how this production system would apply to language learning consider how it might serve to start ACT on the way to the sequencing of words in language. Suppose, ACT hears Mommy utter the phrase "Spot barks." This would be represented as the two events:

(@SAY @MOMMY SPOT)
(@SAY @MOMMY BARKS)

which do occur in close temporal succession. There are numerous intersections between these two events including the terms @SAY and MOMMY. Also there would be the connection that both SPOT and BARKS are words. Another important connection might be the event of Spot barking. All this is enough to allow production INNATE2 to execute. The production it would designate would be:

(@SAY @MOMMY SPOT)
=>
(PREDICT ACT (SAY @MOMMY BARKS))

This is a production which predicts that @MOMMY will say barks after saying Spot. As such the production is both wrong and not particularly useful. We would want the production to be constrained to those situations where Spot is barking. We would like a general production for subject-verb sequences--not one that just applies to Mommy, Spot, and barking. We would like a production that can be used for guiding ACT's behavior as well as predicting others. As we will see this production can evolve in the desired direction through the mechanisms of generalization and discrimination. However, this production is the first step in the direction of the final, desired production.

The productions in Table 4 are not unique to producing linguistic behavior. We speculate that they and others like them can serve as the basis of much of a child's cognitive development. Of course, learning by contiguity and similarity are generally considered to be inadequate to

account for the child's rapid development in language and other areas. There are in ACT additional learning mechanisms of generalization, discrimination, and composition which serve to further direct the course of learning.

5.5 Designation with Substitution

Designation in ACT occurs through its data base. That is, ACT designates certain propositions in memory to serve as patterns for conditions and other propositions in memory to serve as patterns for actions. In some cases, the propositions after which the designated production will be patterned must be built anew. Other times, however, these propositions already exist in memory. So, for instance, the production on p. 52 for describing a ball originated from the actual events of Mommy pointing to a ball and ACT saying "ball." It is often critical to use incoming information from the environment to form production patterns.

However, sometimes the environmental patterns are not exactly what is needed. It is necessary to replace certain nodes in the environmental event by other nodes. One good example of the need for this comes from learning by modelling. Consider the following production which might be useful in learning-by-modelling:

```
(LOOK-AT GVmodel LVevent)
(LVprop = (SAY GVmodel LVphrase))
=>
(CONDITION LVevent)
(ACTION LVprop)
(REPLACE GVmodel ACT)
```

This production encodes that, if the model says a phrase in response to perception of an event, then ACT should build a production in which it does the same. So, suppose the model (GVmodel = Mommy) says "Hi" when she sees a friend wave to her. Thus we have:

(LOOK-AT MOMMY (ALICE WAVE))

and
(SAY MOMMY HI)

which matchs the condition of the modelling production. Then the new production designated will be of the form:

(LOOK-AT ACT (ALICE WAVE))
=>
(SAY ACT HI)

in which ACT has replaced MOMMY wherever it occurred in the original production.

Another place where replacement is useful is in introducing variables into a designated production. Consider, for instance, a student faced with the following definition of CONS in the language LISP:

(CONS A B) = (A . B)

which he represents to himself as:

(EVALUATE (CONS A B) (LP A DOT B RP))

where LP stands for left parenthesis and RP stands for right parenthesis. The task is to translate this information into a production for processing CONS in such a way that the production is not specific to the terms A and B. The following designating production will convert the above definition of CONS into the appropriate production.

```
P1:     (EVALUATE (LVoperation = (LVfunction LVarg1 LVarg2)) LVstruct)
        =>
```

```
(CONDITION LVoperation)
(ACTION LVstruct)
(REPLACE LVarg1 LVx)
(INDEFINITE LVx)
(REPLACE LVarg2 LVy)
(INDEFINITE LVy)
```

This designating production, applied to the above structure, results in the following production:

```
P2:     (CONS LVterm1 LVterm2)
        =>
        (LP-LVterm1-DOT-LVterm2-RP)
```

In matching P1, LVarg1 is bound to A and LVarg2 to B. However, the structure designated has LVarg1 and LVarg2 replaced by nodes LVx and LVy. The INDEFINITE predicates in P1 cause new local variables LVterm1 and LVterm2 to be placed in the designated production rather than any definite nodes. So the substitution mechanism in ACT allows a way for designating productions which are more general than the data they are designated from because additional local variables have been introduced.

6 Production Strengthening

In the ACT system there can be a number of productions which are capable of applying at one time. Some of these productions can be in direct conflict. For instance, the system may have multiple productions encoding alternate rules for pluralizing a noun. Selection among competing productions is partially determined by their strength where the strength of a production is a measure of its past history of success in application.

6.1 Strength Computation

Every time a production is executed and every time another consistent production is designated or executed, the strength of the production is incremented by one unit. Note that the strength of a production increases, not only when it executes, but when a consistent production executes. By "consistent" is meant a production which applies in the same situation and performs the same action. This means that the condition of the executed production must be a special case of the condition of the to-be-strengthened production or identical with it. The actions must be identical (after instantiation of variables). Thus, it is possible to determine whether one production is consistent with another simply by doing a syntactic comparison of the two productions. This principle of consistency places into the ACT system a means by which a more general production will accrue strength more rapidly than its specific variants. This will prove important to understanding ACT's generalization powers.

We have developed a set of rules for calculating production strength which produce behavior on ACT's part which has human-like qualities but does appear to be more rapid. Productions are initially designated with strength .1. Each subsequent strengthening of a production results in an increment of .025. We set the original total strength of the production system at 20. These values undoubtedly tune the system to display learning more rapid than that of humans. However, since the cost of a trial is very expensive computationally in ACT, we are motivated to speed up the rate of learning.

The time to apply a production is an inverse function of its strength relative to the strength of all competing productions selected for possible application on that trial. Thus, if s is the strength of a production and S is the sum of its strength and the strength of all of its competitors, the mean time for that production to apply is cS/s where c is a time scale factor. This predicts that time should decrease with frequency of a production but increase with the number and strength of a production's competitors. Both predictions were confirmed in Anderson (1976, Section 6.3).

6.2 Designation has Precedence over Strength

There needs to be a means of overriding the effects of production strength. Adults can temporarily override very strong rules in response to simple instruction. So, for instance, despite the fact that the "add s" pluralization rule is quite strong, we have little difficulty in instituting a substitute "add er" pluralization rule (e.g., I saw three doger). This simply indicates that top-down rules elicited by deliberate intention take precedence over bottom-up rules elicited in response to data. To put it in ACT terms: If a production is designated it will be given a temporary precedence over any production elicited by the data. To put it in another way: Production strength is only relevant to data-selected productions. ACT always executes designated productions first and then applies data-selected productions according to their strength.

The well-known experiments by LaBerge (1974) provide nice support

for these operating principles in ACT. LaBerge used a mixture of a successive and a simultaneous matching paradigm: The subject always saw a single letter first. Then usually he saw a second letter which he had to identify as same or different from the first. This is the successive case. Sometimes, however, the second thing he saw was a pair of different letters and he had to judge whether these were the same--without regard to the first letter. This is the simultaneous case. In the successive case the subject could designate ahead of time a procedure for recognizing the specific letter tested. In the simultaneous case, however, the correct recognition procedure had to be selected by the data. So, in the first, successive designation case we would not expect strength of a procedure to have an effect. However, in the second simultaneous data-driven case, strength should have an effect. LaBerge manipulated strength of the procedure by manipulating the amount of experience with the to-be-recognized letters. In one case these letters were familiar roman alphabet and in the other case they were unfamiliar, experimenter-created, letter-like symbols. LaBerge found slower recognition times for the weaker, unfamiliar letters only in the simultaneous, data-driven case--as ACT would predict.

We have seen how ACT is able to compensate for the weakness of a production by direct designation. There might seem to be a symmetric possibility--that is, directly designating that strong productions should not apply. However, we think the evidence is that people are very poor at inhibiting well-practiced mental procedures under conditions in which

they normally apply--although people are fairly good at inhibiting overt behavioral indications of a procedure. As an exercise try not comprehending a sentence spoken to you. The only means that seems at all effective (short of plugging your ears) seems to be to set up some competing mental procedure (mental multiplication) to keep oneself from processing the sentence. So we seem not to have the same facility to inhibit a procedure as we have to designate one. Similarly, ACT does not have a symmetric ability to block application of a procedure.

6.3 Interaction Between Strength and Specificity

To fully understand how the strength principle works one must consider its interaction with the principle of specificity. On any cycle of the production system, ACT will attempt to apply a set of productions called the APPLYLIST. The productions can be placed in the APPLYLIST by designation or by data-selection. There are two factors that determine whether a production is data-selected. First, the nodes named in its condition must be active (see page 28). If this factor is satisfied, the strength of a production determines its probability of being placed on the APPLYLIST. However, once on the APPLYLIST, another principle is used to determine which productions will be executed--this is specificity. If there are two productions, one whose condition is more specific than another (i.e., requires additional structure to be in memory) and if all the conditions of the more specific production are met, then the more specific production will apply and will block the execution of its more general counterpart.

To see these two principles at operation, consider the behavior of a mini-simulation of the ACT system learning to refer to objects. We will assume that it initially refers by nouns without determiners. The following production generates this behavior.

```
P1:     (REFER ACT LVobj)
        (CONCEPT-FOR LVobj LVidea)
        (WORD-FOR LVword LVidea)
        =>
        (SAY ACT LVword)
```

However, we assume that at sometime the program notes that determiners are used in model speech, and the use of the determiner is related to whether the listener knows about the object. The program has the following production to monitor what the model says and what the listener knows about the object the model is speaking of:

```
D1:     (LVprop = (REFER GVmodel LVobj))
        (LVprop1 = (CONCEPT-FOR LVobj LVidea))
        (LVprop2 = (WORD-FOR LVword LVidea))
        (LVprop3 = (LVrelation LISTENER LVobj))
        (LVprop4 = (SAY MODEL (LVword1 LVword)))
        =>
        (CONDITION LVprop LVprop1 LVprop2 LVprop3)
        (ACTION LVprop4)
        (REPLACE GVmodel ACT)
        (REPLACE LVobj LVobj1)
        (INDEFINITE LVobj1)
        (REPLACE LVidea LVidea1)
        (INDEFINITE LVidea1)
        (REPLACE LVword LVwordx)
        (INDEFINITE LVwordx)
```

The REPLACE and INDEFINITE commands cause new local variables to be introduced for the nodes bound to variables LVobj, LVword, and LVidea--see earlier discussion on page 57. (Variables could also be introduced

by a slower process of generalizing from a number of examples--see the discussion in the next section.) This production, upon seeing a definite noun phrase, would designate a production of the form:

```
P2:     (REFER ACT LVobjx)
        (CONCEPT-FOR LVobjx LVideax)
        (WORD-FOR LVwordx LVideax)
        (AWARE LISTENER LVobjx)
        =>
        (SAY ACT (THE LVwordx))
```

Upon seeing an indefinite noun phrase, it would designate the production:

```
P3:     (REFER ACT LVobjy)
        (CONCEPT-FOR LVobjy LVideay)
        (WORD-FOR LVwordy LVideay)
        (UNAWARE LISTENER LVobjy)
        =>
        (SAY ACT (A LVwordy))
```

These two productions, P2 and P3, have more specific conditions than the original production P1. Therefore, if on the APPLYLIST, they would apply and block the less correct production P1. However, they are initially much weaker and so have initially a smaller probability of being in the APPLYLIST.

We ran a simulation of the behavior of a production system with productions P1 and D1. Production P1 was given strength 20 but D1 was given a strength of only .1 to reflect its newly created status. We then ran a simulation in which we alternated between model trials and test trials. On model trials we gave the system an example of the model using definite and indefinite articles, alternating between these. On test trials we required the system to generate noun phrases, alternatively to

express definite and indefinite objects. So a series of four trials would be to model definite, test definite, model indefinite, test indefinite. Initially production D1 was unreliably applied, reflecting its weak strength. Similarly, when it did designate productions these productions were weak and applied unreliably. However, gradually D1 built up strength through use and came to behave more reliably. Similarly, the production it designated became more frequent.

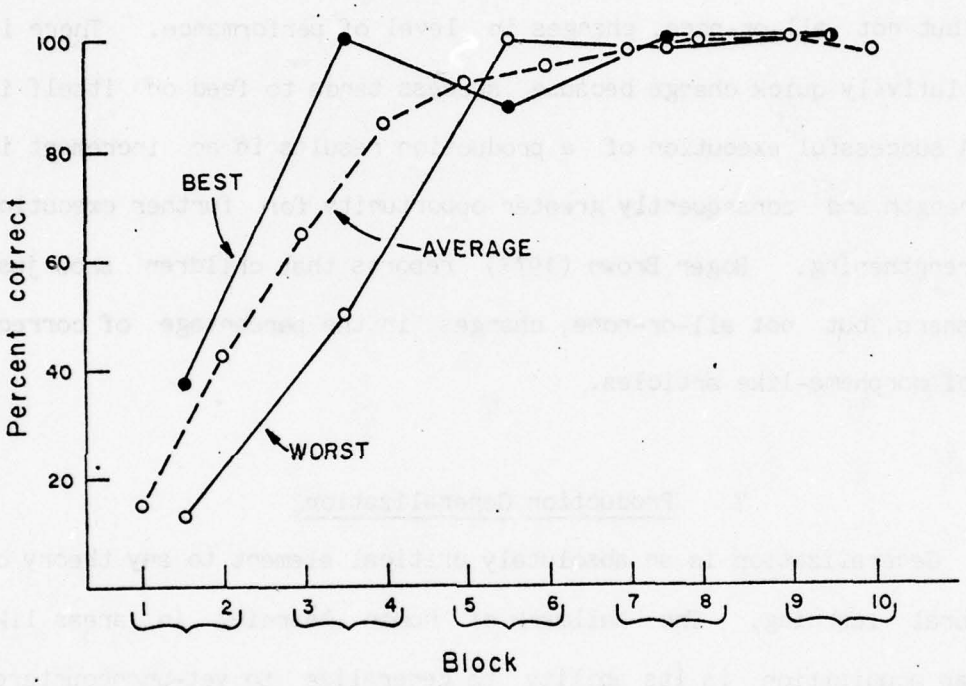


Figure 4. Increase with practice in ACT's correct application of the articles a and the.

We ran ten simulations over forty trials (10 sets of four alternating trials). Figure 4 presents the results of these simulations

in terms of the percentage of correct article usage in each block of 4 trials. There is a gradual increase in the frequency with which the correct forms are used. Data from individual runs displayed much the same behavior. We have reproduced in Figure 4 the best and worst individual learning courses. (In these individual curves we have averaged together two blocks of four.) They also reflect relatively rapid, but not all-or-none, changes in level of performance. There is this relatively quick change because success tends to feed on itself in ACT. A successful execution of a production results in an increment in its strength and consequently greater opportunity for further execution and strengthening. Roger Brown (1973) reports that children show just these sharp, but not all-or-none, changes in the percentage of correct usage of morpheme-like articles.

7 Production Generalization

Generalization is an absolutely critical element to any theory of procedural learning. The hallmark of human learning in areas like language acquisition is its ability to generalize to yet-unencountered examples. It is because of generalization that we are not forced to learn separately how to behave in each possible situation but rather can go from a relatively limited training experience to a competence that is applicable to an unlimited number of situations. It is also failure to produce the appropriate kinds of generalization that served as one of the focal points for the attack of the new psycholinguistics on traditional learning theory.

7.1 Designated Generalizations

Generalization in the ACT framework occurs in one of two ways. First, the initial productions that are designated are somewhat general--as our previous mini-systems clearly illustrate. Suppose the system is learning by modelling its behavior according to examples provided by a model. In failing to encode all the circumstances it is implicitly abstracting and selecting. In failing to specify certain features it is implicitly generalizing. There is also the more explicit form of generalization when a variable is specified in the designating production. This is nicely illustrated in the example involving encoding of the LISP CONS function on page 56. Here ACT is deliberately replacing specific constants like "A" and "B" with variables like LVarg1 and LVarg2.

7.2 Automatic Generalization

The second major type of generalization is a process that occurs automatically without designation. It is the process by which ACT compares a pair of productions, extracts what they have in common, and proposes new productions which will apply in all the circumstances of the original productions. More importantly, this generalized production will apply in situations where neither original production applied. We speculate that this generalization process is particularly important in relatively unstructured situations where there is little direct guidance from instruction. For instance, this learning process would be important

in inducing the principles of program writing from examples or in modelling first language acquisition. This section is mainly concerned with analyzing the automatic generalization process. First, we will define the process by which two productions are generalized. Then, we will consider the factors that determine whether there will be an attempt to generalize two productions and the factors that determine what the fate will be of the generalized production.

7.3 Definition of Generalization

Our definition of generalization can be seen as an adaption of Vere (1977): A production $C_1 \Rightarrow A_1$ is considered a generalization of $C_2 \Rightarrow A_2$ if $C_1 \Rightarrow A_1$ applies in every circumstances that $C_2 \Rightarrow A_2$ does (and possibly others); and in these circumstances $C_2 \Rightarrow A_2$ causes just the same changes to the data base as $C_1 \Rightarrow A_1$. We can specify the conditions under which one production will be a generalization of another: Consider any consistent scheme for renaming local variables and nodes in C_2 by local variables in C_1. We will refer to this as a substitution θ. Let θC_2 denote C_2 after these substitutions have been made. Similarly, let θA_2 denote the action after the same substitutions. Then $C_1 \Rightarrow A_1$ is a generalization of $C_2 \Rightarrow A_2$ if and only if there is some θ such that $C_1 \subset \theta C_2$ and $A_1 = \theta A_2$. Let us apply this definition to an example production for making an inference about geography.

```
P1:     (IN LVplace LVcontinent)
        (WET LVplace)
        (HOT LVplace)
        (FLAT LVplace)
        =>
        (CAN-GROW LVplace RICE)
```

P2: (IN LVlocation ASIA)
(WET LVlocation)
(HOT LVlocation)
(FLAT LVlocation)
(HAS LVlocation ORIENTALS)
=>
(CAN-GROW LVlocation RICE)

Consider the following substitution 0 for production P2: replace ASIA by LVcontinent and replace LVlocation by LVplace. The condition of P2 after this substitution becomes θC_2 =

(IN LVPlace LVcontinent)
(WET LVplace)
(HOT LVplace)
(FLAT LVplace)
(HAS LVplace ORIENTALS)

This is a proper superset of C_1 in that it has the additional clause (HAS LVplace ORIENTALS). Thus, we satisfy the criterion $C_1 \subset C_2$. The action of P_2 after substitution becomes θA_2 = (CAN-GROW LVPlace RICE) which is identical to the action of A_1. Thus, we satisfy the constraint $A_1 = \theta A_2$. So, by definition P_1 is a generalization of P_2. P_2 differs from P_1 in that it will only apply to Asian countries with Oriental inhabitants. We will denote the fact that P_1 is a generalization of P_2 by $P_1 < P_2$.

Note that production P_1 achieves its more general status by two means--both replacement of nodes by variables (in this case ASIA by LVcontinent) and by deletion of clauses. The extremes of these means of generalization are rather bizarre possibilities. One way to have production P1 be more general than P2 is to have P1 consist totally of

variables. However, such a P1 could not be matched. An even more extreme case would be if the condition of P1 had no clauses. This would also be impossible to use. So, when we speak of a production P_1 being more general than another production P_2, we exclude the possibilities that P_1 may consist of all variables or that P_1 has no clauses in its condition.

The automatic generalization routine tries to find common generalizations for a pair of productions P_1 and P_2. A production P_3 is a common generalization of P_1 and P_2 if $P_3 < P_1$ and $P_3 < P_2$. More specifically, ACT tries to form maximal common generalizations. P_3 is a maximal common generalization of P_1 and P_2 if $P_3 < P_1$ and $P_3 < P_2$ and there exists no other P such that $P_3 < P$ and $P < P_1$ and $P < P_2$. A maximal common generalization P_3 is one which deletes the minimal number of clauses in P_1 and P_2 and replaces the minimal number of nodes in P_1 and P_2 by variables. It is possible (although it has not often been the case in our applications) that a pair of productions can have more than one maximal common generalization. In such cases ACT selects one of these arbitrarily.

Before considering further the circumstances under which ACT chooses to form generalizations and how it computes such generalizations, it would be useful to have an example of the program performing generalizations.

7.4 Two Examples of Generalization

Here we would like to work through two examples of the generalization mechanism. The first example that we want to consider is an attempt to learn the sentence frames that define a word like give. The situation modelled in this simulation is one in which ACT is observing a teacher who is uttering sentences and indicating the meaning of these sentences by pointing to events in the world. Table 5 contains the designating production for this example plus the productions designated. Production D is the only designating production needed. The first input to the production was:

```
(SAYS TEACHER (JOHN-GAVE-THE-BALL-TO-JANE))

(POINTS TEACHER (CAUSE-CHANGE JOHN
                (POSSESSION JOHN BALL TIME1)
                (POSSESSION JANE BALL TIME2)))
```

Production D designates the sentence as the condition of a production and the meaning as the action. Thus, it is designating a comprehension production--to map from sentence to meaning. The production so designated is given in Table 5 as G1. Note that the meaning designated does not involve the concept GIVE directly but decomposes it into more primitive concepts of CAUSE CHANGE and POSSESSION. We hardly mean to endorse semantic decomposition (see Anderson, 1976, 73-74 and page 116 of this report for a discussion). However, this example does show ACT has the potential for semantic decomposition.

Table 5
The Designating Productions and Generalizations in the Example Involving Learning the Syntactic Structure of GIVE

```
D:      (SAY TEACHER (LVsentence =
           (LVword1-LVword2-LVword3-LVword4-LVword5-LVword6)))
        (POINTS GVteacher LVprop)
        =>
        (CONDITION LVsentence)
        (ACTION LVProp)

G1:     (JOHN-GAVE-THE-BALL-TO-JANE)
        =>
        (CAUSE-CHANGE JOHN
                (POSSESSION JOHN BALL TIME1)
                (POSSESSION JANE BALL TIME2))

G2:     (BILL-GAVE-THE-DOLLY-TO-MARY)
        =>
        (CAUSE-CHANGE BILL
                (POSSESSION BILL DOLLY TIME1)
                (POSSESSION MARRY DOLLY TIME2))

G3:     (LVagent-GAVE-THE-LVobject-TO-LVrecipient)
        =>
        (CAUSE-CHANGE LVagent
                (POSSESSION LVagent LVobject LVtime1)
                (POSSESSION LVrecipient LVobject LVtime2))
G4:     (MARY-GAVE-TO-JOHN-THE-BALL)
        =>
        (CAUSE-CHANGE MARY
                (POSSESSION MARY BALL TIME5)
                (POSSESSION JOHN BALL TIME6))

G5:     (BILL-GAVE-TO-JANE-THE-DOLLY)
        =>
        (CAUSE-CHANGE BILL
                (POSSESSION BILL DOLLY TIME7)
                (POSSESSION JANE DOLLY TIME8))

G6:     (LVagent1-GAVE-TO-LVrecipient1-THE-LVobject1)
        =>
        (CAUSE-CHANGE LVagent1
                (POSSESSION LVagent1 LVobject1 LVtime5)
                (POSSESSION LVrecipient1 LVobject1 LVtime6))
```

There are admittedly a number of unrealistic simplifications in the example. For instance, the production D matches the whole 6 word string. It would be more realistic to have a number of productions matching a word or two at a time. Also it is a little strange to suppose we have a teacher pointing at an event which spans times in the past. These problems could be remedied but at the cost of considerable complication in the exposition.

The second induced production, G2, results from input of the sentence Bill gave the dolly to Mary. On the basis of these two sentences a generalized production, G3, is produced by replacing with variables those nodes that differ in the two productions. This general production will comprehend any statement of the form Person gave the object to person. Productions G4-G6 reflect a training and generalization history to produce G6 which will handle general statements of the form Person gave to person the object. These two training histories were performed to show that ACT would not confuse the two sentence structures and try to form generalizations of the two. There is no way to substitute variables from the condition of G3 into G6 such that the two actions are the same and this is a necessary condition for generalization. In G3 the fourth word denotes the object in the action, while in G6 it denotes the recipient.

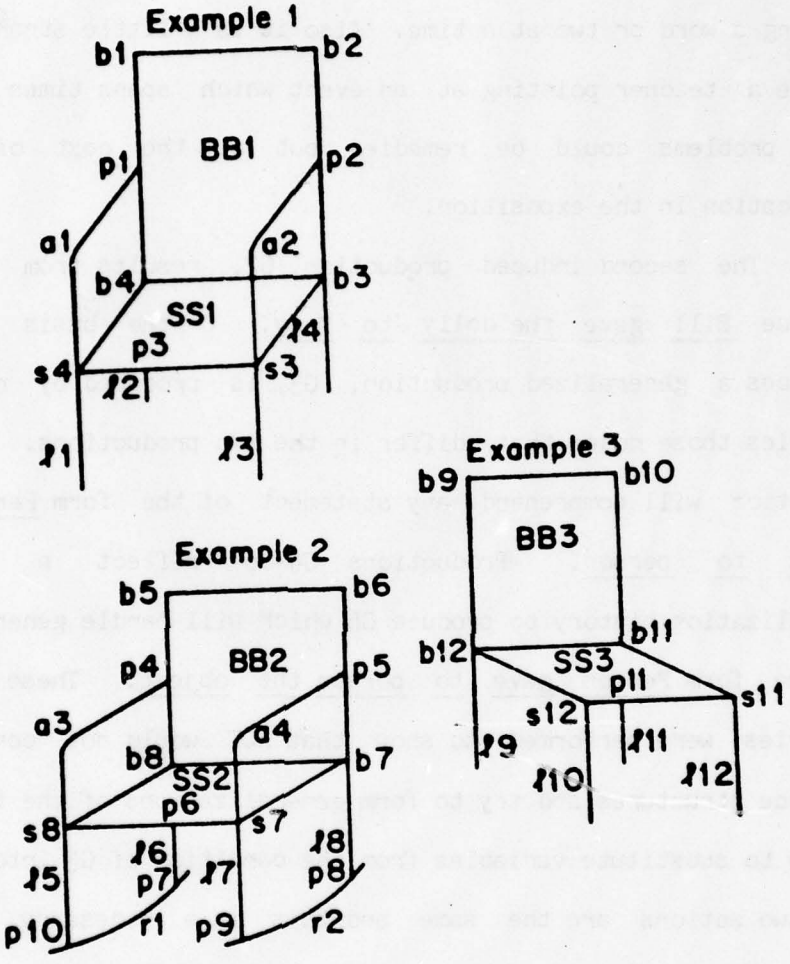


Figure 5. The three examples of chairs presented to the program--adapted from Hayes-Roth & McDermott-1976.

The second example to be discussed concerns the induction of the concept of a chair. (This example has not yet been implemented.) It is

an adaptation of a problem posed and solved in Hayes-Roth and McDermott (1976). Figure 5 illustrates three chairs that will be presented to the system as examples. It is assumed that these examples are encoded by the system as productions, where the condition of the production encodes the description of the object and the action encodes the assertion that the object is a chair.

Table 6
Productions Designated and Generalized in the
Induction of the Concept of Chair

```
E1:     (ARM A1 01 P1 S4)
        (ARM A2 01 P2 S3)
        (LEG L4 01 B3 B3a)
        (LEG L3 01 S3 S3a)
        (LEG L2 01 P3 P3a)
        (LEG L1 01 S4 S4a)
        (QUADRILATERAL BB1 01 B1 B2 B3 B4)
        (QUADRILATERAL SS1 01 B4 B3 S3 S4)
        (ONPOINT BB1 P1)
        (ONPOINT BB1 P2)
        (ONPOINT SS1 P3)
        (BROWN BB1)
        (BROWN SS1)
        =>
        (CHAIR 01)

E2:     (ARM A3 02 P4 S8)
        (ARM A4 02 P5 S7)
        (LEG L8 02 B7 P8)
        (LEG L7 02 S7 P9)
        (LEG L6 02 P6 P7)
        (LEG L5 02 S8 P10)
        (ROCKER R1 02 P7 P10)
        (ROCKER R2 02 P8 P9)
        (QUADRILATERAL BB2 B5 B6 B7 B8)
        (QUADRILATERAL SS2 B8 B7 S7 S8)
        (ONPOINT BB2 P4)
        (ONPOINT BB2 P5)
        (ONPOINT SS2 P6)
        (BROWN BB2)
        (ORANGE SS2)
        =>
        (CHAIR 02)
```

```
E1*E2:  (ARM LVa1 LVo LVp1 LVs4)
        (ARM LVa2 LVo LVp2 LVs3)
        (LEG LV14 LVo LVb3 LVb3a)
        (LEG LV13 LVo LVs3 LVs3a)
        (LEG LV12 LVo LVp3 LVp3a)
        (LEG LV11 LVo LVs4 LVs4a)
        (QUADRILATERAL LVbb LVo LVb1 LVb2 LVb3 LVb4)
        (QUADRILATERAL LVss LVo LVb4 LVb3 LVs3 LVs4)
        (ONPOINT LVbb LVp1)
        (ONPOINT LVbb LVp2)
        (ONPOINT LVss LVp3)
        (BROWN LVbb)
        =>
        (CHAIR LVo)

E3:     (LEG L9 03 B12 B12a)
        (LEG L12 03 S11 S11a)
        (LEG L11 03 P11 P11a)
        (LEG L10 03 S12 S12a)
        (QUADRILATERAL BB3 03 B9 B10 B11 B12)
        (QUADRILATERAL SS3 03 B12 B11 S11 S12)
        (ONPOINT SS3 P11)
        (BROWN SS3)
        (ORANGE BB3)
        =>
        (CHAIR 03)

(E1*E2)*E3:
        (LEG LV14 LVo LVb4 LVb4a)
        (LEG LV13 LVo LVs3 LVs3a)
        (LEG LV12 LVo LVp3 LVp3a)
        (LEG LV11 LVo LVs4 LVs4a)
        (QUADRILATERAL LVbb LVo LVb1 LVb2 LVb3 LVbx)
        (QUADRILATERAL LVss LVo LVbx LVb3 LVs3 LVs4)
        (ONPOINT LVss LVp3)
        =>
        (CHAIR LVo)
```

Consider production E1 in Table 6 which encodes the first example. It describes the object 01. Interpreting the clauses in the condition it asserts:

1. 01 has arm A1 with termini P1 and S4.

2. 01 has arm A2 with termini P2 and S3.

3. 01 has leg L4 with termini B3 and B3a. (I have taken the liberty of labelling the bottoms of the legs in the Hayes-Roth & McDermott figures.)

4. 01 has leg L3 with termini S3 and S3a.

5. 01 has leg L2 with termini P3 and P3a.

6. 01 has leg L1 with termini S4 and S4a.

7. 01 has a quadrilateral-shaped area, BB1, defined (clockwise from upper left quadrant) by B1 B2 B3 and B4.

8. 01 has a quadrilateral-shaped area, SS1, defined by B4 B3 S3 S4.

9. Point P1 is on BB1.

10. Point P2 is on BB1.

11. Point P3 is on SS1.

12. BB1 is brown.

13. SS1 is brown.

Note that this is an encoding of the line drawing without the benefit of three-dimensional information--the seat and back are defined as quadrilaterals not as rectangles, it is not encoded that leg L2 joins at B4 but only that it is stops at P3 where it is occluded by the seat SS1.

Table 6 also contains production encodings of the other two examples. It is assumed that these productions represent the output of a perceptual analysis of the picture. We do not actually have such perceptual analyzers. Like Hayes-Roth and McDermott we directly give the program these productions.

Table 6 contains E1*E2, the generalization that was formed after presentation of the first two examples. In producing this generalization, ACT has replaced all the "constants" that differ between E1 and E2 by variables. Since the two examples disagree as to whether the seat is brown, that particular feature has been deleted from the concept. Finally, the two rocker descriptions from E2 are not represented. E3 is an example which identifies that the arms are not necessary to the chair concept. Also the chair is presented at a different angle and so the right back leg is occluded. Finally, this example establishes that the seat need not be brown. Production (E1*E2)*E3 displays the results of generalizing E1*E2 with E3. It specifies the following conditions of an object (LVo) to be considered a chair.

1. LVo has leg LV14 with termini LVb4 and LVb4a.

2. LVo has leg LV13 with termini LVs3 and LVs3a.

3. LVo has leg LV12 with termini LVp3 and point LVp3a.

4. LVo has leg LV11 with termini LVs4 and LVs4a.

5. LVo has a quadrilateral-shaped area, LVbb, defined by points LVb1 LVb2 LVb3 LVbx.

6. LVo has a quadrilateral-shaped area, LVss, defined by points LVbx LVb3 LVs3 LVs4.

7. The occlusion point LVp3 is on LVss.

The above almost perfectly defines the standard perspective view of a standard chair. The one thing missing is that the back leg LV14 (the one not occluded) is attached at point LVb4 to either LVb3 or LVbx at the back of the seat. This is something that cannot be encoded easily in a single production but would require a pair of productions--one for each possibility.

ACT, fed a long series of chair examples, would develop a family of such productions--some more general and some more specific--recall that ACT does not replace a specific production by a generalized version. Both the specific and the general production coexist. It has been argued for such concepts (Rosch & Mervis, 1975; Wittgenstein, 1953) that there does not exist a single set of features to define chair but a family of feature sets. This family of productions produced by ACT would nicely correspond to this notion of a family of concepts.

7.5 The Problem of Efficiency

It turns out that a number of other researchers (e.g., Hayes-Roth & McDermott, 1977 and Vere, 1977) have been working on generalization routines that operate in similar contexts. While our own work evolved independently, it is also the case that this other work does have claim

to historical precedence. These enterprises do differ in the computational techniques by which they attempt to discover the generalizations. Our program uses a rather brute force technique of trying to put clauses into correspondence by substitution of variables. Clauses which cannot be put into correspondence are deleted. We achieve efficiency by heuristics for ordering how clauses are set into correspondence. If there are n clauses in the condition of P_1 and m clauses in P_2 ($n > m$) there are potentially $n! / (n-m)!$ possible correspondences to consider. It is therefore wise to try to make these correspondences in a way that maximizes how quickly a good correspondence is considered and quickly identifies a bad generalization. We also gain certain efficiencies because of restrictions, shortly to be discussed, in the generalization process.

It has been observed by Hayes-Roth (1977) that this generalization problem in its most general form is an NP-complete problem, and therefore probably has no uniformly computationally satisfactory solution. It is widely believed that the time to solve NP-complete problems will be an exponential function of problem complexity. Therefore, it was necessary to restrict our generalization routine to computing only so long before giving up in trying to find a generalization. The reason that tractable solutions can occur in applied problems is that these applied problems are not random selections from the general generalization problem, but tend to have certain features which can be capitalized upon by heuristics. Our heuristics for ordering

matching of clauses are an attempt to capture these constraints. Since ACT is a psychological simulation these constraints amount to psychological claims about the kind of generalizations humans will make. However, we have not yet seen a way to make empirical tests of these claims. We omit specification of them in the interest of avoiding excessive technical detail and because of an inability to provide empirical interpretation.

7.6 Focusing of the Generalization Process

There is a major aspect of the generalization problem which is addressed in the ACT program which is not addressed in other efforts. This is the problem of focus: ACT as a realistic system would contain hundreds of thousands of productions. How does ACT decide upon which ones it should try to generalize? It would be disastrous to attempt to generalize all possible pairs of productions. Not only would this be astronomically costly, it also would produce many spurious generalizations.

ACT restricts its generalizations to those initiated by newly designated productions. That is, when a production is designated ACT attempts to generalize it with other productions that it has. It can be shown that this restriction to newly formed productions does not miss any potential generalizations. Unfortunately, this restriction by itself does not produce sufficient computational savings. However, ACT only tries to generalize designated productions with productions that are on

the APPLYLIST. Given that their conditions are active, productions are placed on the APPLYLIST with a probability dependent on their strength. This activation-plus-strength criterion means that generalization will be restricted to productions that are relevant to the current context and which have had a fair history of past success.

Even with these restrictions there is the danger of spurious generalizations. Consider the following pair of productions, one of which might be on the APPLYLIST and the other newly created

```
P1:     (IN LVlocation ASIA)
        (WET LVlocation)
        (HOT LVlocation)
        (FLAT LVlocation)
        =>
        (CAN-GROW RICE LVlocation)

P2:     (HAVE LVlocation roads)
        (IN LVlocation Vietnam)
        (NEAR LVlocation RIVER)
        ABS (IN LVlocation mountains)
        =>
        (CAN-GROW RICE LVlocation)
```

Production P1 is a rule about conditions that favor rice-growing in Asia while P2 is a rule that predicts rice-growing areas in Vietnam. We would not want the following generalization to form:

```
P3:     (IN LVlocation LVplace)
        =>
        (CAN-GROW RICE LVlocation)
```

which is the maximal common generalization of P1 and P2. To avoid such obviously spurious generalizations we restrict generalization so that it is only able to delete so many clauses. If n is the number of clauses in

the smaller of the two to-be-generalized conditions, then ACT must keep at least .75n clauses in the generalization. The only restriction on replacement of nodes by variables is that all nodes cannot be replaced.

7.7 Overgeneralization

Of course, it is not possible to avoid all spurious generalizations. Given that humans overgeneralize, it is desirable that the ACT program overgeneralize in similar circumstances. For instance, children learning their first language (and it also appears adults learning a second language--see Bailey, Madden, & Krashen, 1974) overgeneralize morphemic rules. Thus, a child will generate mans, gived, etc. We are happy to report ACT does the same. (see Section 8.2)

Overgeneralizations require correction. If the generalization is just plain wrong, then it will be taken care of by ACT's strength mechanisms. It is initially designated with little strength and will not come to apply unless it reliably designates correct behavior. However, it can be the case, as it is with morphemic overgeneralization, that the general rule is close to correct but needs some tuning. This tuning is the responsibility of the discrimination processes which are our next topic of discussion.

8 Discrimination

Whether by direct designation or by automatic generalization, ACT can generate too-general productions. Correction of these productions

depends on a discrimination process. Productions can be directly designated which are more discriminate than existing ones. This is like the ability to directly designate very general productions. However, there is also an automatic facility for discrimination. As was the case with generalization, designated discrimination is postulated to be more important in highly structured learning situations where ACT can intelligently go about debugging its errors. Automatic discrimination is more important in less structured situations. The direct designation option does not involve any possibilities that we have not already considered. Therefore, this section will focus on the automatic discrimination process.

The current automatic discrimination mechanism requires that there be the potential for negative feedback to the system about its behavior. ACT keeps track of which productions have created which memory structures. If it is decided that a memory structure is bad, then the production which created that structure is punished. The decision that a structure is bad can come from direct feedback from a teacher, comparison with a model, computations that note a contradiction, and presumably other sources as well. If a production is punished its strength is reduced by a factor of 1/4 in the current implementation. It is also the case that punishment evokes the automatic discrimination process -- as we will discuss shortly.

8.1 Discrimination by restriction versus discrimination by exception

It is useful to separate conceptually two ways in which discrimination can cure problems of overgeneralization. There can be overgeneral rules which fail to specify a feature of the problem to which they apply. For instance, we might have the following production for deciding about rice growing:

```
(WET LVlocation)
(HOT LVlocation)
=>
(CAN-GROW RICE LVlocation)
```

This rule requires an additional condition clause to restrict the inference to flat areas. The other possible way a production may be overgeneral is that there may be exceptions to production rules. To cure this overgeneralization it it necessary to characterize the features of the situations which are exceptions, rather than the features of the situations where the rule holds.

ACT's automatic solution to both of these discrimination problems is a process called variant-spawning. A variant of a production is spawned by generating a new production which has some further information specified in its condition. There are two types of variants that can be created. First, a production can be designated which calls for the same behavior, but which has a more specific set of conditions. This is the way that productions are created to correct under-restricted rules such as that used in the rice growing example. Suppose this discriminant

production more accurately characterizes the correct situation than its more general source. Then it will apply in all the situations where the more general production would have applied correctly. Therefore, it will be strengthened in all the situations that the source is strengthened, but will avoid the punishment that the source gets for misapplication. The second type of discrimination is to designate a production with a different response and more discriminant condition. This is the way to encode exceptions to a rule. If this discriminated production is sufficiently strengthened it will take precedence, when applicable, over the more general rule because of ACT's specificity ordering. It will be useful to have terms for these two types of discrimination. The first we will refer to as discrimination-by-restriction and the latter as discrimination-by-exception.

Note that in encoding exceptions the discriminant production only takes precedence ovfr the general. The general production continues to operate when the discriminant version is not applicable. This is because except in the circumstance caught by the discriminant production, the general production works just fine and is strengthened. In contrast, in discrimination-by-restriction the discriminant production comes to replace the overgeneral production.

8.2 An Example Requiring Discrimination and Generalization

To help firm up these various concepts it will be useful to have an example at hand. We chose a mini-simulation concerned with the

acquisition of verb inflections for tense and subject noun inflection for number. Table 7 shows the four designating productions that we used for this example. Production N1 responds to the appearance of a string of morphemes spoken by a model. It sets attention (GVtok) to the beginning of the string. Productions N2 and N3 are responsible for scanning through the string. Production N2 deals with words or morphemes that have connections to concepts--(WORD-FOR LVidea LVword)--while N3 deals with morphemes that lack such a connection (i.e., inflections). Production N2 attaches a global variable GVholdc to the propositions that encode the word-idea connection. As we will see, these propositions will be made part of the condition of the designated production. Production N4 is elicited after N2 and N3 have completed their scan of the string. This is signalled by setting GVtok to the last morpheme token which allows the condition (SAY MODEL (...-GVtok)) to match. The embedded structure (...-GVtok) is shorthand for a string whose last element is GVtok. The condition of N4 also requires a match to the meaning intended by the model--a relation (LVrelation) between two objects (LVagent and LVobject) at a specific time (LVtime). The other two propositions in the condition encode the connection between the two objects and class names for these objects (e.g., between a boy and the class of boys). The production designated in the action includes the model's meaning (LVprop2), the connections between objects and their class concepts (LVprop3 and LVprop4), and the word-idea connections associated with GVholdc. The action of the designated production is the sentence spoken

AD-A052 671 YALE UNIV NEW HAVEN CT DEPT OF PSYCHOLOGY F/G 5/10
A THEORY OF THE ACQUISITION OF COGNITIVE SKILLS.(U)
FEB 78 J R ANDERSON , P J KLINE, C M BEASLEY N00014-77-C-0242
UNCLASSIFIED NL

2 OF 2
AD
A052671

END
DATE
FILMED
5-78
DDC

by the model. In this designation ACT replaces reference to the model by reference to itself.

Table 7

A Production Set for Learning
Main Clause Structure Plus Number Inflection

```
N1:     (SAY MODEL (LVtok))
        =>
        (GVtok = LVtok)

N2:     (GVtok-LVtok)
        (LVmorph LVtok)
        (LVprop = (WORD-FOR LVidea LVmorph))
        =>
        (GVtok = LVtok)
        (GVholdc LVprop)

N3:     (GVtok-LVtok)
        (LVmorph LVtok)
        ABS (WORD-FOR LVidea LVmorph)
        =>
        (GVtok = LVtok)

N4:     (LVprop1 = (SAY MODEL (...-GVtok)))
        (LVprop2 = (MEANS MODEL (LVrelation LVagent LVobject LVtime)))
        (LVprop3 = (CLASS-FOR LVagent LVidea1))
        (LVprop4 = (CLASS-FOR LVobject LVidea2))
        =>
        (CONDITION LVprop2 LVprop3 LVprop4 GVholdc)
        (ACTION LVprop1)
        (REPLACE MODEL ACT)
        (UNBIND GVholdc GVtok)
```

As an example of the productions designated suppose the model says the sentence The boy likes the girl and we assume ACT understands what the model means by reference to the situation in which the sentence is uttered. The input to these productions will be:

```
(MEANS MODEL (@LIKE PER1 PER2 TIME1))
(CLASS-FOR PER1 @BOY)
(CLASS-FOR PER2 @GIRL)
(SAY MODEL (THE-BOY-LIKE-S-THE-GIRL))
```

The designated production would be:

```
(MEANS ACT (@LIKE PER1 PER2 TIME1))
(CLASS-FOR PER1 @BOY)
(CLASS-FOR PER2 @GIRL)
(WORD-FOR @BOY BOY)
(WORD-FOR @GIRL GIRL)
(WORD-FOR @LIKE LIKE)
=>
(SAY ACT (THE-BOY-LIKE-S-THE-GIRL))
```

which, for further discussion, we will abbreviate as

```
P1:     (MEANING LIKE BOY GIRL TIME1)
        =>
        (THE-BOY-LIKE-S-THE GIRL)
```

Other similar productions would be designated such as:

```
P2:     (MEANING KICK GIRL DOG TIME2)
        =>
        (THE-GIRL-KICK-S-THE-DOG)
```

From such pairs the following generalization would be made:

```
P3:     (MEANING LVverb LVsubj LVobj LVtime)
        =>
        (THE-LVsubj-LVverb-S-THE-LVobj)
```

Based on sentences such as:

```
        The dogs chase the cat
        The dog chased the cat
        The dogs chased the cat
```

the following generalizations would result:

```
P4:     (MEANING LVverb LVsubj LVobj LVtime)
        =>
        (THE-LVsubj-S-LVverb-THE-LVobj)

P5:     (MEANING LVverb LVsubj LVobj LVtime)
        =>
        (THE-LVsubj-LVverb-ED-THE-LVobj)

P6:     (MEANING LVverb LVsubj LVobj LVtime)
        =>
        (THE-LVsubj-S-LVverb-ED-THE-LVobj)
```

Productions P3-P6 deal with the cases of singular and plural subject, present and past tense verb. These four productions all have the same condition, implying that the choice of inflection is arbitrary. Of course, this is incorrect--the productions are too general. The conditions do not include tests for number of subject or for time: This is because the original designating productions in Table 7 did not consider this information relevant.

The following are the four discriminant versions of these productions that we would like ACT to have:

```
P7:     (MEANING LVverb LVsubj LVobj LVtime)
        (PRESENT LVtime)
        (SINGULAR LVsubj)
        =>
        (THE-LVsubj-LVverb-S-THE-LVobj)

P8:     (MEANING LVverb LVsubj LVobj LVtime)
        (PRESENT LVtime)
        (PLURAL LVsubj)
        =>
        (THE-LVsubj-S-LVverb-THE-LVobj)

P9:     (MEANING LVverb LVsubj LVobj LVtime)
        (PAST LVtime)
        (SINGULAR LVsubj)
        =>
        (THE-LVsubj-LVverb-ED-THE-LVobj)

P10:    (MEANING LVverb LVsubj LVobj LVtime)
        (PAST LVtime)
        (PLURAL LVsubj)
        =>
        (THE-LVsubj-S-LVverb-ED-THE-LVobj)
```

8.3 Example Continued: Discrimination by Variant Spawning

ACT tries to discover the more discriminant variants required to

correct P3-P6 by inspecting the contexts in which the overgeneral productions P3-P6 apply successfully. It trier to find constraints on the bindings of the local variables at these points of successful application. Constraints are any additional propositions in which the local variable bindings occur. So, for instance, it finds that when P3 applied LVtime was present and LVsubj was singular. These constraints need to be added to P3 to create the production P7. When it finds the correct constraints, the discriminate production will gradually take over for its parent, being reinforced in the same situations as the parent production, but avoiding the punishment that the parent gets.

An early version of the discrimination process spawned such variants every time a production successfully applied. A proposition was selected at random from among the propositions, if any, and was attached to the bindings of the local variables. If there were no such propositions, beyond those already specified as the production condition, no variant was spawned. This process tended to spawn a great many harmless but useless production variants. They were harmless because they were as at least as correct as their parents since they applied in no new contexts, they were useless when there was no problem with the behavior of their parents. When there was a problem with the parent behavior, chances were that the discriminations were the wrong ones and some other variant should have been spawned. It is true that with enough time this process should work to solve ACT's discrimination problems. However, the method seemed unacceptably inefficient. In line with the

conjecture discussed earlier about the relationship between AI considerations and psychological considerations, we decided to jettison this particular mode of discrimination. It was very inefficient as AI and, therefore, seemed unacceptable as a psychological theory.

The current discrimination mechanism is much more selective. It will only be evoked if the output of a production is punished. The following production monitors for negative feedback from the model:

```
(SAY MODEL (BAD LVsentence))
=>
(BAD LVsentence)
```

If the model greets the output of the sentence The boy hits the girl with disapproval because the tense should have been past, the condition of this production will be matched and the sentence flagged as bad. This will cause the production, P7, which generated the sentence to be punished. The variable bindings are stored for the application of P7 which led to this punished result. When the production applies next and is not punished , the variable bindings of this application are compared to the bindings of the misapplication. This identifies which variables were differently bound in the two circumstances. A search is made for some proposition which is true of the current bindings but not the misapplication bindings. One such proposition is selected at random for spawning a variant. (The fact that just one is designated means a pair of discriminative refinements is necessary to add two propositions as in P7). In the current application, no such constraining proposition is found, and one of the local variables is replaced by one of the bindings that differs from the bindings in the misapplication.

Note that for P7 to have been punished it must have been selected--indicating that it is probably fairly strong. This means that discrimination is limited to fairly strong productions which have already had some history of success. It would be wasteful to discriminate weak productions which might be so wrong that all effort at correction would be wasteful.

This discriminant production no more replaces its parent than does a generalized production replace the productions that gave rise to it. Rather, the new production is spawned in addition to the parent. Whether it will come to replace the parent because of accrual of strength depends on its track record of success and failure.

So, discrimination proceeds by intelligently trying to constrain the variables in an overgeneral production. As a direct complement to generalization, discrimination can proceed by adding additional clauses to a condition or by replacing variables with nodes.

Productions P7-P10 are not quite adequate in themselves. They do not deal with irregular past tenses. For instance, to deal with <u>hit</u> we need the following production:

```
P11:    (MEANING HIT LVsubj LVobj LVtime)
        (PAST LVtime)
        (SINGULAR LVobj)
        =>
        (THE-LVsubj-HIT-THE-LVobj)
```

This production would be generated, with the condition clause (PAST LVtime), at an intermediate stage of generalization from examples. If selected, P11 will apply rather than the general P9 because P11 is more

specific. By the same means we would get intermediate states of generalization involving regular verbs like kicked:

```
P12:    (MEANING KICK LVsubj LVobj LVtime)
        (PAST LVtime)
        (SINGULAR LVobj)
        =>
        (THE-LVsubj-KICK-ED-THE-LVobj)
```

As P12 is correct there is no problem with having it as well as the redundant P9 which generalizes over the verb. Thus, in ACT there may be a number of regular verbs with their own productions to handle them as well as for irregular verbs. For a special encoding of a verb to be effective it would be necessary that it occur with sufficient frequency to build up adequate strength. This accounts for the observation that only high frequency words can maintain irregular forms. Low frequency words cannot have a strong special form and must depend on the general production.

The fact that there can be redundant special coding of regular words accounts for the fact that subjects can more rapidly produce regular forms of words than irregular when the words are of equal frequency (McKay, 1976). This is because there are two productions--the general and the specific that can produce the regular inflection but only one production for the irregular inflection.

8.4 Effect of Punishment on Learning

It is worthwhile to consider what this discrimination mechanism says about the effectiveness of punishment or negative information. In

the ACT model it is not possible to simply punish and "stamp out" a negative behavior. It seems a fair generalization from the learning literature (e.g., Estes, 1970; Hilgard & Bower, 1974) that it is not possible to do this with living organisms either. An undesired production in ACT can be gotten rid of in basically one way--by strengthening a production which will compete with it. For instance, we saw punishment work with an overgeneral production--but punishment worked by forcing the designation of new, discriminative productions that attempted to characterize what separated the circumstances of successful application of the production from the circumstances of non-successful application. Punishment is only effective in that it is a stimulus for discrimination. It remains for successful applications and consistent designations (or reinforcement, if you will) to build up the strength of the correctly discriminated production.

It is a widely held premise in the first language learning literature that negative information is not at all effective. ACT is not in total agreement with this point of view. It would certainly be the case that telling a child a sentence is wrong would be useless. A great many productions go into a sentence's generation and a child would face an impossible task trying to guess which one or ones were to blame. However, more focused negative feedback should be helpful. For instance, if a child utters: two foots and is corrected: two feet he might be able to recognize that the inflection of foot is at fault. If so, he could punish the production responsible for insertion of the 's plural for foot

and designate a new production for feet. However, Cazden (1964) reports no advantage of providing a child with such correction over providing him with an opportunity for general interaction in the language. However, this result is somewhat in dispute (McNeil, 1970). In any case, ACT is on record to the effect that sufficiently focused correction on generating errors should produce improvement.

9 Production Composition

9.1 Definition of Composition

Production composition is the newest learning mechanism that we have projected for ACT (it is not implemented yet). Our interest in production composition was stirred by the work of Clayton Lewis and our ideas about it largely come from him. The basic idea behind production composition is quite simple. Suppose we have productions P1: C_1 => A_1 and P2: C_2 => A_2 and it is observed that these productions tend to occur in sequence. Then we can form the composite:

P3: C_1 & (C_2 - A_1)
=>
A_1 & A_2

where (C_2 - A_1) denotes all the conditions in C_2 that were not created in the action of A_1. Thus, P3 applies in just the circumstance where P1 and P2 would apply and it adds to memory the effects of both P1 and P2--i.e., the actions A1 and A2. However, what once took two steps now takes one step.

Consider this composite-forming possibility as it might be applied to a language processing example. Suppose initially the system had the productions:

```
L1:     (THE GVtok)
        (BEFORE GVtok LVtok)
        =>
        (DEFINITE GVtop)
        (GVtok = LVtok)

L2:     (LVword GVtok)
        (ADJECTIVE LVword)
        (WORD-FOR LVword LVidea)
        (BEFORE GVtok LVtok)
        =>
        (LVidea GVtop)
        (GVtok = LVtok)

L3:     (LVword GVtok)
        (NOUN LVword)
        (WORD-FOR LVword LVidea)
        =>
        (CLASS-OF GVtop GVidea)
```

This is a three production sequence for analyzing determiner-adjective-noun phrases. In this example it is necessary to make a distinction between words and tokens of these words as they appear in sentences. The construction (THE GVtok) indicates GVtok is an instance of THE, and the construction (LVword GVtok) indicates GVtok is an instance of LVword. The production L1 processes the definite article the; L2 processes adjectives and predicates their meaning of the noun phrase topic, GVtop; and L3 parses nouns and builds a structure to indicate that the topic is an instance of that noun. By a process of composing L1 with L2 and that composite with L3 (or by composing L2 with L3 and that composite with L1) we would get:

```
L4:     (THE GVtok)
```

```
(BEFORE GVtok LVtok)
(BEFORE LVtok LVtok*)
(LVword LVtok)
(LVword* LVtok*)
(WORD-FOR LVword LVidea)
(WORD-FOR LVword* LVidea*)
=>
(DEFINITE GVtop)
(LVIDEA GVtop)
(CLASS-OF GVtop LVidea*)
(GVtok = LVtok*)
```

which processes the determiner-adjective-noun sequence.

There are a number of important functions that are served by composition. First, it serves to reduce the number of steps involved in performing a computation. Second, it reduces the total number of condition elements that need to be matched and so speeds up the match.

9.2 Einstellung Effect

The composition of productions can lead to the Einstellung effect. The Einstellung effect refers to the fact that practice on one method of problem solution will produce a blindness to another method of solution.

Consider the following example of Einstellung as studied by Luchins (1942). Subjects are given three water jugs of specified size and are instructed to fill jugs, empty jugs, and transfer water from one jug to another to achieve a desired quantity of water. The following are examples of problems given by Luchins to his subjects. They are given in the form of the capacities of three jugs A, B, and C and the goal amount:

	A	B	C	GOAL
1	21	127	3	100
2	14	163	25	99
3	18	43	10	5
4	9	42	6	21
5	23	49	3	20
6	15	29	3	18

Note that the first four problems are all susceptible to the solution B-A-2C. After working on a number of problems with this same solution subjects tended to become fixated on it. They used this solution on problem 5 although it had the easier solution A-C, which subjects tend to find when they have not had the prior experience. Also many subjects failed to solve problem 6 although it has the simple solution A + C.

Table 8 contains a set of productions for solving waterjug problems. Table 8 contains many details not necessary for understanding the following discussion of composition. Production P1 will try filling empty jars. Its first condition (GOAL LVquantity) is a test for a goal flag; the second condition (LVprop = (CONTAINS LVjar 0)) tests for an empty jar; the third condition (CAPACITY LVJar LVamount) serves to bind the jar's capacity to the variable LVamount; the fourth absence condition prevents the move of dumping a jar just after filling it and thus avoids a foolish loop. The first action of P1 (FILL ACT LVjar LVnext) calls for the fill action; the second action flags the contents of LVjar as no longer empty, the third action encodes the new amount in the jar, and the final action sets the global variable GVlast to an indicant of this fill action which is now the last action of the system.

Table 8
A Production Set for Water Jug Problems

```
P1:     (GOAL LVquantity)
        (LVprop = (CONTAINS LVjar  0))
        (CAPACITY LVjar  LVamount)
        ABS (DUMP ACT LVjar GVlast)
        =>
        (FILL ACT LVjar LVnext)
        (FALSE LVProp)
        (CONTAINS LVJar LVamount)
        (GVlast = LVnext)

P2:     (GOAL LVquantity)
        (LVprop = (CONTAINS LVjar LVamount))
        ABS (FILL ACT LVjar GVlast)
        =>
        (DUMP ACT LVjar LVnext)
        (FALSE LVProp)
        (CONTAINS LVjar 0)
        (GVlast = LVnext)

P3:     (GOAL LVquantity)
        (LVprop1 = (CONTAINS LVjar1 LVamount1))
        (LVprop2 = (CONTAINS LVjar2 LVamount2))
        (CAPACITY LVjar2 LVamount3)
        (GREATER LVamount2 LVamount3)
        (PLUS LVamount1 LVamount2 LVamount5)
        ABS (LVamount1 = 0)
        =>
        (POUR ACT LVjar1 LVjar2 LVnext)
        (CONTAINS LVjar1 0)
        (CONTAINS LVjar2 LVamount5)
        (FALSE LVprop1)
        (FALSE LVprop2)
        (GVlast = LVnext)
```

```
P4:     (LVprop1 = (CONTAINS LVjar1 LVamount1))
        (LVprop2 = (CONTAINS LVjar2 LVamount2))
        (CAPACITY LVjar2 LVamount3)
        (PLUS LVamount2 LVamount4 LVamount3)
        (PLUS LVamount4 LVamount5 LVamount1)
        ABS (LVamount2 = LVamount3)
        =>
        (POUR ACT LVjar1 LVjar2 LVnext)
        (CONTAINS LVjar1 LVamount5)
        (CONTAINS LVjar2 LVamount3)
        (FALSE LVprop1)
        (FALSE LVProp2)
        (GVlast = LVnext)

P5:     (LVprop = (GOAL LVquantity))
        (CONTAINS LVJar LVquantity)
        =>
        (DONE LVprop)

PX:     (LVprop = (GOAL LVquantity))
        (LVprop1 = (CONTAINS LVjar1 0))
        (LVprop2 = (CONTAINS LVjar2 0))
        (LVprop3 = (CONTAINS LVjar3 0))
        (CAPACITY LVjar1 LVamount1)
        (CAPACITY LVjar2 LVamount2)
        (CAPACITY LVjar3 LVamount3)
        (PLUS LVamount1 LVamount4 LVamount2)
        (PLUS LVamount3 LVamount5 LVamount4)
        (PLUS LVamount3 LVquantity LVamount6)
        =>
        (FILL ACT LVjar2 LVnext1)
        (POUR ACT LVjar2 LVjar1 LVnext2)
        (POUR ACT LVjar2 LVjar3 LVnext3)
        (DUMP ACT LVjar3 LVnext4)
        (POUR ACT LVjar2 LVjar3 LVnext5)
        (CONTAINS LVjar2 LVquantity)
        (CONTAINS LVjar1 LVamount1)
        (CONTAINS LVjar3 LVamount3)
        (DONE LVprop)
        (FALSE LVProp1)
        (FALSE LVprop2)
        (FALSE LVProp3)
```

Production P2 will dump the contents of a jar. Production P3 encodes the action of emptying one jar into another. Production P4 encodes the operation of emptying as much of one jar into another as there is room. Production P5 recognizes when the goal has been achieved. This set of five productions is adequate to solve the waterjug problems at hand and to make the desired point about the relationship between production composition and the Einstellung effect.

This production set randomly tries various movements until it finds the correct sequence. In the case of problems like 1 through 4 the correct sequence would be, starting with empty jugs: Apply P1 to B, P4 to A and B, P4 to C and B, P2 to C, and P4 to C and B. Repetition of this series leads to composition of this series of productions into larger units. Production Px represents the outcome of the composition of P1, P4, P4, P2, P4, and P5 to solve the first four problems. It would spell out the steps in the solution to problems 1-5 in one action. Applied to problem 5 it would block (because of the specificity principle) the simpler sequence of productions possible: P1 to A and then P4 to A and C. Strengthening this sequence would interfere with the perception of a solution to problem 6: P1 to A, P3 to A and B, P1 to C, and P3 to B and C. There are two reasons for this: First, the correct sequence would be interfered with by practice of the incorrect sequence because of the role of relative strengths in production selection (see the equation on p. 59). Second, subsequences of the one coded in Px would match this problem description, be more specific than any of the productions P1-P5, and so be applied rather than the correct production.

9.3 Conditions for Evoking Composition

It remains to specify the circumstances under which composition will occur in ACT. With each priming problem that ACT is given, the productions in Table 8 will be applied. At first their order of application will be haphazard but they will always conclude with the sequence P1, P4, P4, P2, P4, P5. ACT will keep track of which pairs of productions follow which others in close succession and will form composites of such pairs. Thus it will compose P1 and P4 into a production which we will call P6, P4 and P2 into P7, and P4 and P5 into P8. Once P6, P7, P8 are formed and strengthened they will tend to apply as a sequence. Then P6 and P7 can be composed into P9 and P9 and P8 into PX given in Table 8. We have restricted composition to operate on pairs of productions for simplicity sake.

This composition works to create a production like PX which encodes the correct sequence of steps for problems like 1-5. However, composition would also work on other regularities in the haphazard applications of P1-P4 before the correct sequence. Thus, ACT may attempt to pour A and then C into B. If it tries this move frequently it will be embodied by a composite. The difference between such composites and PX is that they do not encode a problem solution. In contrast, PX does encode a problem solution because it includes P5 which signals a solution. It is a deficit of the current ACT system that after forming the composite PX, it cannot give priority to this production in light of the fact its action includes a final solution to the problem. We will

discuss a remedy for this system defect in the last section. Thus, we are only capable of partially producing Einstellung in ACTF by means of composition. We can form the composite but we are not able to assign it the priority that it demands.

10 Significant Problems and Future Directions

We feel quite optimistic about the potential of the learning theory that has been outlined. The mechanisms proposed seem both psychologically valid and computationally powerful. We are currently performing empirical tests of the implications of the learning theory. What pleases us most is the apparent generality of these mechanisms--that they seem applicable to such a wide range of cognitive procedures. The obvious test of the computational power is to try the ACT system out on some large scale learning example. It would be worthwhile to review the major obstacles to doing this and the methods we have in mind for overcoming these obstacles. Discussion of these issues of course also identifies the future direction for research in the ACT project.

However, before doing so we would like to acknowledge that the basic learning mechanisms we have been proposing--designation, strengthening, generalization, discrimination, and composition are not very different from the ideas that have been offered in traditional learning theory. One might wonder why the now-classic criticisms of learning theory (e.g., Anderson, 1976; Anderson & Bower, 1973; Bever, Fodor & Garrett, 1968; Chomsky, 1959 1965; Katz & Postal, 1964) do not

apply. They do not apply because the basic production system mechanism is much more powerful as a performance model than the basic mechanisms (e.g., the S-R bond) of the traditional theories. Criticisms of traditional theories really rested on the computational inadequacy of the performance mechanisms.

10.1 Difficulties with Matching

The current matching scheme is one in which a subset of productions are selected by the activation process and then the conditions of each of those are independently matched against memory. There are at least three serious difficulties with this scheme:

i. Redundant Matching Many selected productions will share large portions of their condition pattern, but these common patterns must be rematched independently for each production. It would be preferable to have a scheme for matching these common subparts once and for all.

ii. Complex Conditions on Instantiations The nature of ACT matching is to retrieve a single instantiation for a production's condition and not to retrieve all possible instantiations. This makes it difficult to express certain complex conditions. Consider the following case: One of the simple things we want our LISP learner to do is to apply the syntactic definition of LISP to the recognition of certain LISP structures. So, suppose ACT is trying to determine whether a certain expression--call it EXP, is an S-expression. It has two relevant rules:

Rule a If an expression is an atom it is an S-expression, represented (IMPLIES ATOM S-EXPRESSION)

Rule b If an expression is a dotted-pair it is an S-expression, represented (IMPLIES DOTTED-PAIR S-EXPRESSION)

It represents its goal as: GOAL1 = (GOAL (S-EXPRESSION EXP)). The node GOAL1 stands for this high level goal. Suppose ACT has tried out both definitions for S-expression and they have failed. This is represented:

```
(TRIED-OUT GOAL1 RULEa)
(TRIED-OUT GOAL1 RULEb)
```

We want to write a production condition that would recognize that all the rules relevant to proving S-expression have been exhausted and that ACT should flag this goal as a failure:

```
(LVgoal = (GOAL (LVprop LVexp)))
ABS ((LVrule = (IMPLIES LVprop1 LVprop))
     ABS (TRIED-OUT LVgoal LVrule))
=>
(FAILURE LVgoal)
```

This production asserts that if one's goal is to show property LVprop and there are no rules for establishing LVprop which have not been already tried out, then flag that subgoal as a failure. This condition cannot be matched in ACT because of the double embedding of absence tests required to reflect the double negation. This would require one to retrieve and inspect all instantiations of (IMPLIES LVprop1 LVprop)--something which the current match scheme does not do.

iii. Partial Matching The third difficulty with the current matching algorithm is that it does not permit partial matches to be detected. There are a number of circumstances where partial matching is

essential. One of these is in generalization--where a partial match must be detected between the conditions of two productions. This is currently performed in ACTF by mechanisms other than the general matcher. However, it is probably the case that one match procedure should both detect overlap between two conditions and detect overlap between a condition and data. It is also the case that we want to detect partial overlap between two sets of data to be able to model how subjects detect the similarities between two concepts. We also want to be able to detect differences. This is particularly true in a problem-solving situation where one wants to adopt difference-reduction techniques. For instance, our definition of a dotted pair is: a left parenthesis followed by an S-expression followed by a dot followed by an S-expression followed by a right parenthesis. We might have as a goal to show that STRUCTURE A is a dotted pair and our description of STRUCTURE A is Left parenthesis followed by Structure B followed by a dot followed by Structure C followed by a right parenthesis. Comparing this to our general definition of a dotted pair and noting the differences we would set up the subgoals of showing that Structure B was an S-expression and Structure C was an S-expression.

Partial matching seems pervasive in learning and other aspects of human cognition. The correspondence made between a production condition and data in the current ACT is a complete match--if anything fails to correspond the entire match fails. We have yet to convince ourselves that in this testing of production conditions matching should ever be

partial. There is too much potential for ACT to lose control of the processing and produce bizarre results. However, it seems that ACT should be able to match conditions to conditions and data to data and in these circumstances there should be the potential for partial matching.

10.2 Data-Flow Matching

There seems to be a single solution to these three problems with matching. This is to use the idea of data flow matching such as described by Forgy (1977) for the Carnegie-Mellon production system.

The basic idea behind data-flow matching is fairly simple, although the implementation details seem quite complex. As we have not passed the implementation hurdle, we will just describe the basic idea. Figure 6 shows an example of the matching net that might be used. The terminal nodes of this net can be assumed to refer to simple one-proposition patterns. In Figure 6 there are six such patterns. Each pattern can appear in the condition of a number of productions. Figure 6 shows how they might underlie the five productions in Table 9.

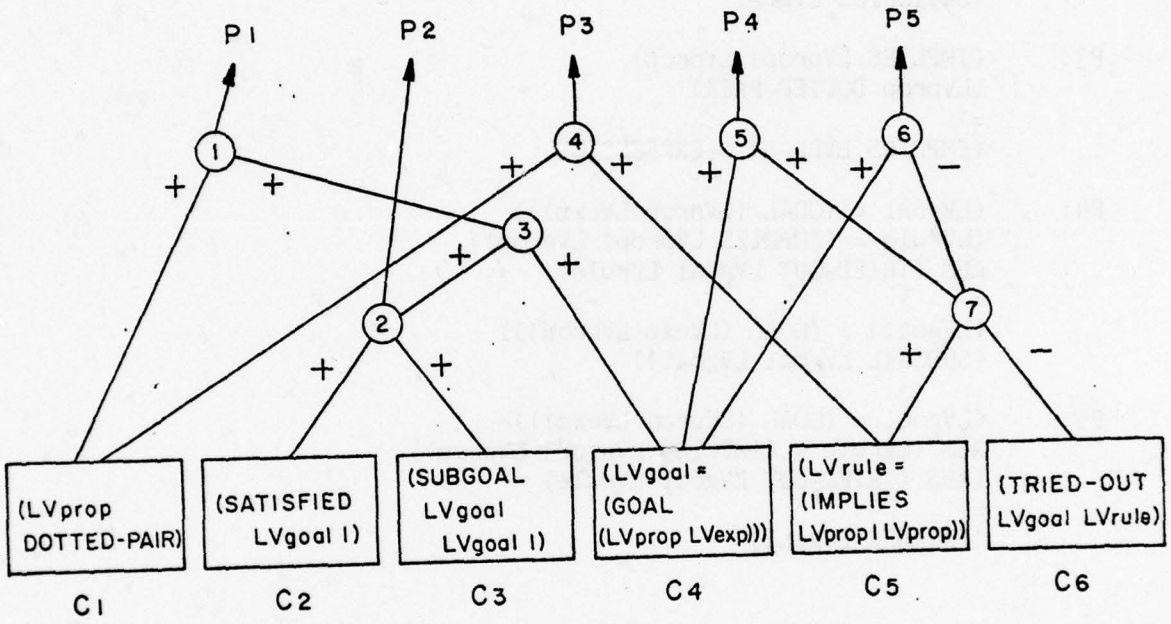


Figure 6. A data-flow network combining overlapping conditions in five productions.

Table 9

The Productions Represented in the Data Flow Network of Fig. 6

```
P1:     (LVgoal = (GOAL (LVprop LVexp)))
        (LVprop DOTTED-PAIR)
        (SUBGOAL LVgoal LVgoal1)
        (SATISFIED LVgoal1)
        =>
        (S-EXPRESSION LVexp)

P2:     (SUBGOAL LVgoal LVgoal1)
        (SATISFIED LVgoal1)
        =>
        (SATISFIED LVgoal)

P3:     (IMPLIES LVprop1 LVprop)
        (LVprop DOTTED-PAIR)
        =>
        (IMPLIES LVProp1 S-EXPRESSION)

P4:     (LVgoal = (GOAL (LVprop LVexp)))
        (LVrule = (IMPLIES LVprop1 LVprop))
        ABS (TRIED-OUT LVgoal LVrule)
        =>
        (LVgoal1 = (GOAL (LVexp LVprop1))
        (SUBGOAL LVgoal LVgoal1)

P5:     (LVgoal = (GOAL (LVpred LVexp)))
        ABS (LVrule = (IMPLIES LVprop1 LVprop))
        (ABS (TRIED-OUT LVgoal LVrule)
        =>
        (FAILURE LVgoal)
```

These productions are purely to illustrate the matching network. They are somewhat artificial and somewhat simplistic productions for reasoning about LISP expressions. P1 asserts that if the goal is to show LVexp has some property LVprop that is equivalent to being a dotted pair, and its subgoal is satisfied (forgetting that there may be more than one subgoal) then LVexp is an S-expression. P2 asserts that if a subgoal is satisfied the goal is satisfied. P3 asserts that if LVprop1 implies a property, LVprop, equivalent to being a dotted-pair, then LVprop1 also implies the property of being an S-expression. P4 asserts that if the goal is to show LVexp is LVprop, and there is a rule that LVprop1 implies LVprop and this rule has not been tried out, then set as a subgoal to show LVexp is LVprop1. Finally, P5 asserts that if the goal is to show LVexp is LVprop and there are no more unused relevant rules, then this goal has failed. This last production has the problematic condition considered earlier (p. 108).

The patterns are matched just once at the terminal nodes. All possible instantiations can, in principle, be retrieved. At the higher nodes the instantiations from a number of subnodes are combined under the constraint that they be compatible. So, for instance, consider node 2 which combines C2--(SATISFIED LVgoal1)--with C3--(SUBGOAL LVgoal LVgoal1). C2 matches any two element structure whose first element is SATISFIED. C3 matches any three-element structure whose first element is SUBGOAL. However, their combination at node 2 in the net enforces the constraint that the second element of C2 be identical with the third

element of C3. This serves to filter out many possible instantiations. For this reason, the higher nodes in the matching network are called filters. It is referred to as a data flow matching procedure because data flows upward through the network filters.

Lower elements can flow either positively or negatively into a filter. When a positive and a negative element meet at a filter, the filter only accepts instantiations of the positive elements which are not compatible with any instantiations of the negative elements. Consider the problematical production P5. The lower filter, 7, finds all instantiations of (LVrule = (IMPLIES LVprop1 LVprop)) that are not compatible with (TRIED-OUT LVgoal LVrule). That is to say, the value assigned by C5 to LVrule does not agree with any value assigned to LVrule by C6. Then the higher filter, 6, finds all instantiations of C4= (LVgoal = (GOAL LVprop LVexp)) that are incompatible with what is passed up by filter 7. That is, no instantiation of LVprop passed up from 7 agrees with the instantiation passed up from C4. If there are any such instantiations that pass filter 6, P5 will apply.

So, the data flow matcher allows us to compute more complex conditions. It also has the obvious virtue of avoiding redundant matching of condition propositions. Since each element of a production is separately matched, there is a potential for partial matching. We could simply pass up a count of the number of condition propositions matched and the number mismatched. The data-flow scheme also has the potential for implementing top-down selection as to which productions

will be matched. It is possible to focus processing on certain productions by sending signals down the matching network from these productions enabling data to flow up from the condition nodes that support these productions while data cannot flow from other condition nodes. This ability for top-down focusing promises additional efficiency and power.

This matching proposal has a host of experimental predictions. It makes predictions about what should be the optimal way to structure condition tests in the matching tree and what sort of experience should create these structures. It makes a number of predictions about effects of availability of data (as influenced by frequency, recency, and availability of information) that are in contrast to predictions of the earlier ACT model (see Anderson, 1976; Ch. 8). It makes interesting predictions about interactions between top-down and bottom-up priming influences. It also makes some interesting predictions about partial matching results. We are setting out to test these various predictions.

10.3 Knowledge Representation

Knowledge representation is another point at which the weaknesses of ACTF are becoming apparent. The reader familiar with earlier descriptions of ACT will notice a switch in this paper from a HAM-like structure to a relation-plus-argument structure. In actual fact, ACTF is implemented with the HAM knowledge syntax, but we have chosen the new syntax for this paper in part because we judge it to be more conducive to

exposition. It is also the case that this is the type of network syntax being projected for the new version of ACT. We are planning a representational switch not just to facilitate communication but also because it appears to be more efficient for matching than ACT's former subject predicate structures. ACT's current representation uses a number of levels of network structures within a single proposition. This means that it is expensive to search a proposition to reject it as false. It is also the case that after seven years of research we know of no data that strongly supports these within-proposition distinctions made by the HAM representation.

Another feature needed to make ACT's representation more efficient for search is to use a less homogeneous set of structures and nodes. The fewer structural distinctions there are, the fewer distinct nodes, the more one structure looks like the next and the harder it is to find the structure searched for. As noted by Hayes-Roth (1977) and Hayes-Roth and Hayes-Roth (1977) this fact is an argument against attempts to decompose meaning representation into homogeneous structures of a few semantic primitives (e.g., Norman & Rumelhart, 1975; Schank, 1975). Rather than representing terms like give, take, beg, borrow, trade, lend, etc. at a low level where they are very similar and hence very confusable, we want to represent these terms at a high level where the confusability does not exist.

In fact, it seems that an important dimension of learning is for ACT to discover and use high-level labels for frequently reappearing

subconfigurations of patterns. So for instance, ACT should, and we suppose a child does, create a concept like throw after encountering a repeating sequence of someone holding an object, putting his hand behind his back, moving it rapidly forward, releasing the object, and the object moving rapidly as a projectile. Then, ACT could reason about such events simply as a throw and not as the more complex sequence which is confusable with grasp, lift, move, wave, and many other actions.

The extraction and labelling of common subpatterns is hardly a behavior restricted to children. The great deal of recent work on prototype extraction (Bransford & Franks, 1971; Posner & Keele, 1970; Reed, 1972; Neuman, 1974; Rosch & Mervis, 1975; Hayes-Roth & Hayes-Roth, 1977a) gives ample testimony to the pervasiveness of this phenomenon in adult learning. We also feel that it is an important component of learning in the domains of language acquisition and mathematical reasoning. In language learning such pattern extraction is critical to identifying syntactic categories, particularly in languages (e.g., Finnish, Latin) where there are numbers of declensions and conjugations which obey different syntactic rules. In mathematical reasoning, it is critical to develop large subconcepts (e.g., proof by induction, recursion, integration by parts) of what are relatively complex operations. Once these mathematics patterns are formed, it is possible to reason about them as single objects. Indeed, the reasons for pattern extraction or chunking include all the advantages which were identified by Miller in 1956 and documented many times subsequently (Mandler, 1967; Bower & Springston, 1970; Bower, 1975).

The operations by which patterns are extracted are quite similar to the operations by which production conditions evolve--i.e., generalization, discrimination, composition, and strengthening. This points to the conclusion that the distinction between ACT's declarative component (network) and its procedural component (productions) should be weakened or eliminated. That is to say, the operations defined just on the procedural component in ACTF seem also appropriate for the declarative component.

It also seems that processes limited in the current ACT to the declarative component are appropriate for the procedural. We have done a number of experiments now displaying short-term memory priming effects for procedures. It also seems that one of the principal features of productions in ACTF, their non-inspectability, will have to be changed. We are running into many situations where we cannot model human reasoning or human learning without having the contents of one production open to inspection by another production. That is, we need to allow one production to specify in its condition aspects of another production. The need to do this has become particularly apparent in the LISP learning simulation. Here, in order to be able to plan solutions to problems, one has to reason about what his available procedures can do. Again, the example we have worked on concerns the recognition of LISP expressions. Consider a top-down recognition strategy which subjects can implement. There exists the following production which will recognize a dotted pair:

```
(LVobj = (LP-LVtok1-DOT-LVtok2-RP))
(S-EXPRESSION LVtok1)
(S-EXPRESSION LVtok2)
=>
(DOTTED-PAIR LVobj)
```

If there exists an object which entirely satisfies this definition, it can be recognized immediately as a dotted pair. However, it is frequently going to be the case that the nodes matching LVtok1 and LVtok2 are not flagged as S-expressions. What needs to be done is to set up subgoals of proving these to be S-expressions. To do this, ACT must be able to inspect this production, recognize that the action is relevant to the goal of proving something to be a dotted-pair, recognize the condition is partially matched, and set up the appropriate subgoals. The only way we can do this in the current implementation is to encode the information redundantly in productions and network form. However, it would seem better if the production could simply be treated as data.

We noted a similar problem in our simulation of the water jug problem. Here we wanted to give special priority to productions which contained a complete solution. This requires inspecting the action of the production and noting that the action contained the solution. It also requires some way to schedule the priority assigned to a production which we think we can do by the top-down influences permitted in a data-flow matcher.

Another reason for having procedural knowledge available for inspection is so as to be able to model the debugging process. The idea that procedures can be deliberately inspected for "bugs" and corrected has been popularized by a number of AI researchers (Brown, Burton, Hausmann, Goldstein, Huggins, & Miller 1977; Goldstein, 1974; Sussman,

1975). We do not think debugging is typical of most human learning because it requires a considerable expertise in the area to be debugged. We would assign most of human error correction to the discrimination process. However, it does seem undeniable that experts are able to inspect their procedures, find bugs, and correct them. An example in hand is learning to program in the language C where all indexing is initiated at 0 rather than 1 as is typical of a programming language. JRA can remember very systematically reworking procedures for searching arrays, looping, etc. to take account of this fact. This clearly requires that procedures for programming be inspectable. As an aside, it is worth noting that JRA still occasionally slips into the wrong habits even after this conscious correction. This is a nice example that stronger, general procedures can overwhelm the weaker, specific, debugged procedures.

The ACT book (Anderson, 1976) considered eliminating the difference between productions and data, but rejected the possibility. One reason for rejecting that possibility was simply an inability to see in detail how to make the equation between productions and the propositional network. At that time we were focused on a suggestion by Newell to treat each link in the declarative network as an independent production. Now, however, the proposal is to treat a production as a data structure on the same status as a proposition. It does not seem that there will be the same difficulties of coherence with this proposal.

Another basis for objection was the observation that certain

highly overlearned procedures (e.g., knowledge of how to speak a language) do not seem to be inspectable as data. We still do not know what to make of this fact. It still may serve as a basis for a different qualitative treatment of highly overlearned productions. However, the qualitative distinction may not be between declarative and procedural but between intermediate degrees of overlearning versus very high degrees of overlearning.

10.4 Final Remarks about Projected Changes

It is noteworthy that these projected changes in the ACT system will change its performance characteristics at least as much as its learning characteristics. This is despite the fact that the need for such changes became apparent in modelling learning. This reinforces our earlier remarks (see p. 15) about the relationship between a theory of learning and a theory of performance. It is generally acknowledged that the design of a performance system will have strong influences on the learning system. That is, our learning principles will be strongly influenced by our conception of what the end product of the learning process is like. On the other hand, it is also the case, as just illustrated, that work with a learning theory will affect the performance theory. There is a complex and intimate relationship between the two. It is preferable, and fortunately it is possible for us, to pursue both endeavors in parallel.

Footnotes

1. This research is supported by grants N00014-77-C-0242 from the Office of Naval Research and NIE-G-77-0005 from the National Institute of Education.

2. Throughout this section and subsequent sections we will be using a knowledge representation somewhat different than the HAM-based representation used in earlier publications on ACT. The linearization of this notation also avoids many of the infix operators previously developed. The actual implementation of ACTF is still HAM-based with infix linearizations. The current notation, which is more relational and predicate based, is being used because we feel it eases difficulties of exposition. As we discuss in the last section (p. 115) for various reasons this new notation is targeted for later versions of the ACT system. In many other ways we have taken liberties in simplifying, for exposition purposes, the structure of ACT network and productions. For instance, nodes and variables in structures built by ACT productions are often shown with mnemonic labels rather than the nonsense labels actually assigned. Also technical details often essential to actual successful implementation have been omitted. It is our judgment that the simplifications help ease the communication problem without losing anything essential about the important conceptual points. However, for any readers interested in such details, we have available actual listings of the exact mini-systems implemented and of their performance.

3. The language analysis reflected in this production system and some of the other production systems in this paper is quite rudimentary. It would greatly complicate the production system to provide a more accurate model of language analysis. Since the purpose of this system is to illustrate the effect of instruction and not to provide a complete language analysis model, we regard these simplifications as justified.

4. There are certain constraints on this INTERSECTING condition as that it cannot pass through nodes like AT-TIME.

11 References

Anderson, J.R. Language acquisition by computer and child. Technical Report No. 55, Human Performance Center, University of Michigan, 1974.

Anderson, J.R. Computer simulation of a language acquisition system: A first report. In R.L. Solso (Ed.), Information Processing and Cognition: The Loyola Symposium, Hillsdale, N.J.: Erlbaum Assoc., 1975.

Anderson, J.R. Language, memory, and thought. Hillsdale, N.J.: Erlbaum Assoc., 1976.

Anderson, J.R. Induction of augmented transition networks. Cognitive Science, 1977, 1, 125-158.

Anderson, J.R. Computer simulation of a language acquisition system: A second report. In D. LaBerge & S.J. Samuels (Eds.), Perception and comprehension, Hillsdale, N.J: Erlbaum Assoc., 1978.

Anderson, J.R. & Bower, G.H. Human associative memory, Washington, D.C.: Hemisphere Press, 1973.

Anderson, J.R. & Kline, P. Design of a production system. Paper presented at the Workshop on Pattern-Directed Inference Systems. In Sigart Newsletter, June, 1977.

Anderson, J.R., Kline, P. and Lewis, C. A production system model for language processing. In P. Carpenter & M. Just (Eds.), Cognitive Processes in Comprehension. Hillsdale, N.J.: Lawrence Erlbaum Assoc.

Bailey, N., Madden, C., & Krashen, S.D. Is there a "natural sequence" in adult second language learning. English Language Institute and Linguistics Department, Queens College and the Graduate Center, City University of New York, 1974.

Bever, T.G., Fodor, J.A., & Garrett, M. A formal limitation of associationism. In T.R. Dixon & D.L. Horton (Eds.), Verbal Behavior and General Behavior Theory. Englewood Cliffs, N.J.: Prentice-Hall, 1968.

Bower, G.H. Cognitive psychology: An introduction. In W.K. Estes (Ed.), Handbook of learning and cognitive processes, Vol. 1, Hillsdale, N.J.: Erlbaum Assoc., 1975.

Bower, G.H. & Springston, F. Pauses as recoding points in letter series. Journal of Experimental Psychology, 1970, 83, 421-430.

Braine, M.D.S. On learning grammatical order of words. Psychological Review, 1963, 70, 323-348.

Bransford, J.D. & Franks, J.J. The abstraction of linguistic ideas. Cognitive Psychology, 1971, 2, 331-350.

Brown, J.S., Burton, B.R., Hausmann, C., Goldstein, I., Huggins, B. Miller, M. Aspects of a theory for automated student modelling. BBN Report No. 3549, 1977.

Brown, R. A first language. Cambridge, Mass.: Harvard University Press, 1973.

Cazden, C.G. Environmental assistance to the child's acquisition of grammar. Unpublished Ph.D. dissertion, Harvard University, 1965.

Chomsky, N. Verbal behavior (a review of Skinner's book). Language, 1959, 35, 26-58.

Chomsky, N. Aspects of the theory of syntax. Cambridge, Mass.: MIT Press, 1965.

Chomsky, N. & Miller, G.V. Introduction to the formal analysis of natural languages. In R.D. Luce, R.R. Bush, & E. Galanter (Eds.), Handbook of mathematical psychology II. New York: Wiley, 1963.

Estes, W.K. Learning theory and mental development. New York: Academic Press, 1970.

Forgy, C.L. A production system monitor for parallel computers. Technical Report, Computer Science Department, Carnegie-Mellon University, 1977.

Goldstein, I.P. Understanding simple picture programs. AI-TR-294. MIT A.I. Laboratory, 1974.

Hayes-Roth, B. & Hayes-Roth, F. The prominence of lexical information in memory representations of meaning.

Journal of Verbal Learning and Verbal Behavior, 1977, 16, 119-136. (a)

Hayes-Roth, B. & Hayes-Roth, F. Concept learning and the recognition and classification of exemplars. Journal of Verbal Learning and Verbal Behavior, 1977, 16, 321-338. (b)

Hayes-Roth, F. The role of partial and best matches in knowledge systems, Rand Corporation, P-5802, 1977.

Hayes-Roth, F. & McDermott, J. Learning structured patterns from examples. Proceedings of the Third International Joint Conference on Pattern Recognition, 1976, 419-423.

Hilgard, E.R. & Bower, G.H. Theories of learning, New York: Appleton-Century-Crofts, 1966.

Huessmann, L.R. & Woocher, F.D. Probe similarity and recognition of set membership: A parallel-processing serial-feature-matching model. Cognitive Psychology, 1976, 8, 124-162.

Katz, J.J. & Postal, P.N. An integrated theory of linguistic descriptions. Cambridge: MIT Press, 1964.

Kline, P. & Anderson, J.R. The ACTE user's manual. Department of Psychology, Yale University, 1976.

LaBerge, D. Attention and the measurement of perceptual learning. Memory and Cognition, 1973, 1, 268-276.

Luchins, A.S. Mechanization in problem solving. Psychological Monographs, 1942, 54, No. 248.

MacKay, D.G. On the retrieval and lexical structure of verbs. Journal of Verbal Learning and Verbal Behavior, 1976, 15, 169-182.

McNeil, D. The acquisition of language. New York: Harper & Row, 1970.

Mandler, G. Organization and memory. In K. W. Spence and J.A. Spence (Eds.), The Psychology of Learning and Motivation. Vol. 1, New York: Academic Press, 1967, 328-372.

Neumann, P.G. An attribute frequency model for the

abstraction of prototypes. Memory and Cognition, 1974, 2, 241-248.

Newell, A. A theoretical exploration of mechanisms for coding the stimulus. In A. W. Melton & E. Martin (Eds.), Coding processes in human memory, Washington, D.C.: Winston, 1972.

Newell, A. Production systems: Models of control structures. In W.G. Chase (Ed.), Visual Information Processing. New York: Academic Press, 1973.

Norman, D.A., Rumelhart, D.E. & the LNR Research Group. Explorations in Cognition. San Francisco: Freeman, 1975.

Posner, M.I. & Keele, S.W. Retention of abstract ideas. Journal of Experimental Psychology, 1970, 83, 304-308.

Reed, S.K. Pattern recognition and categorization. Cognitive Psychology, 1972, 3, 382-407.

Rosch, E. & Mervis, C.B. Family resemblances: Studies on the internal structure of categories. Cognitive Psychology, 1975, 7, 573-605.

Rumelhart, D.E. & Norman, D.A. Accretion, tuning, and restructuring: Three modes of learning. CHIP Technical Report 63, University of California, San Diego, LaJolla.

Rychener, M.D. & Newell, A. An instructible production system: Basic design issues. In D.A. Waterman & F. Hayes-Roth (Eds.), Pattern directed inference systems, New York: Academic Press, in press.

Schank, R.C. Conceptual information processing. Amsterdam: North-Holland, 1975.

Sternberg, S. Memory scanning: Mental processes revealed by reaction time experiments. American Scientist, 1969, 57, 421-457.

Sussman, G.J. A computer model of skill acquisition. New York: American Elsevier Publishing Co., 1975.

Townsend, J.T. A note on the identifiability of parallel and serial processes. Perception and Psychophysics, 1971, 10, 161-163.

Townsend, J.T. Issues and models concerning the processing of a finite number of inputs. In B.H. Kantowitz (Ed.), Human information processing: Tutorials in performance and cognition. Hillsdale, N.J.: Erlbaum Assoc., 1974.

Vere, S.A. Induction of concepts in the predicate calculus. Proceedings of the Fourth International Joint Conference on Artificial Intelligence, Tbilisi, USSR, 1975, 281-287.

Vere, S.A. Inductive learning of relational productions. Proceedings of the Workshop on Pattern-Directed Inference Systems, Hawaii, 1977(a).

Vere, S.A. Induction of relational productions in the presence of background information. Proceedings of the Fifth International Joint Conference on Artificial Intelligence, Boston, 1977, 349-355 (b).

Weissman,C. LISP 1.5 primer, Belmont, Ca.: Dickenson, 1967.

Winston, P.H. Learning structural descriptions from examples. MIT Artificial Intellligence Laboratory Project AI-TR-231, 1970.

Wittgenstein, L. Philosophical investigations. New York: MacMillan, 1953.

Navy

4 DR. JACK ADAMS
OFFICE OF NAVAL RESEARCH BRANCH
223 OLD MARYLEBONE ROAD
LONDON, NW, 15TH ENGLAND

1 Dr. Jack R. Borsting
Provost & Academic Dean
U.S. Naval Postgraduate School
Monterey, CA 93940

1 DR. JOHN F. BROCK
NAVY PERSONNEL R& D CENTER
SAN DIEGO, CA 92152

1 Dept. of the Navy
CHNAVMAT (NMAT 034D)
Washington, DC 20350

1 Chief of Naval Education and
Training Support)-(01A)
Pensacola, FL 32509

1 CAPT. H. J. CONNERY
NAVY MEDICAL R&D COMMAND
NNMC
BETHESDA, MD 20014

1 Dr. Charles E. Davis
ONR Branch Office
536 S. Clark Street
Chicago, IL 60605

1 Mr. James S. Duva
Chief, Human Factors Laboratory
Naval Training Equipment Center
(Code N-215)
Orlando, Florida 32813

4 Dr. Marshall J. Farr, Director
Personnel & Training Research Programs
Office of Naval Research (Code 458)
Arlington, VA 22217

1 DR. PAT FEDERICO
NAVY PERSONNEL R&D CENTER
SAN DIEGO, CA 92152

Navy

1 CDR John Ferguson, MSC, USN
Naval Medical R&D Command (Code 44)
National Naval Medical Center
Bethesda, MD 20014

1 Dr. Dexter Fletcher
Navy Personnel Research and Development
San Diego CA 92152

1 Dr. John Ford
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. Eugene E. Gloye
ONR Branch Office
1030 East Green Street
Pasadena, CA 91101

1 CAPT. D.M. GRAGG, MC, USN
HEAD, SECTION ON MEDICAL EDUCATION
UNIFORMED SERVICES UNIV. OF THE
HEALTH SCIENCES
6917 ARLINGTON ROAD
BETHESDA, MD 20014

1 Dr. Norman J. Kerr
Chief of Naval Technical Training
Naval Air Station Memphis (75)
Millington, TN 38054

1 Dr. Leonard Kroeker
Navy Personnel R&D Center
San Diego, CA 92152

1 Dr. James Lester
ONR Branch Office
495 Summer Street
Boston, MA 02210

1 Dr. William L. Maloy
Principal Civilian Advisor for
Education and Training
Naval Training Command, Code 00A
Pensacola, FL 32508

1 Dr. Sylvia R. Mayer (MCIT)
HQ Electronic Systems Div.
Hanscom AFB
Bedford, MA 01731

Navy

1 Dr. James McBride
Code 301
Navy Personnel R&D Center
San Diego, CA 92152

2 Dr. James McGrath
Navy Personnel R&D Center
Code 306
San Diego, CA 92152

1 DR. WILLIAM MONTAGUE
NAVY PERSONNEL R& D CENTER
SAN DIEGO, CA 92152

1 Commanding Officer
Naval Health Research
Center
Attn: Library
San Diego, CA 92152

1 CDR PAUL NELSON
NAVAL MEDICAL R& D COMMAND
CODE 44
NATIONAL NAVAL MEDICAL CENTER
BETHESDA, MD 20014

1 Library
Navy Personnel R&D Center
San Diego, CA 92152

6 Commanding Officer
Naval Research Laboratory
Code 2627
Washington, DC 20390

1 JOHN OLSEN
CHIEF OF NAVAL EDUCATION &
TRAINING SUPPORT
PENSACOLA, FL 32509

1 Office of Naval Research
Code 200
Arlington, VA 22217

1 Scientific Director
Office of Naval Research
Scientific Liaison Group/Tokyo
American Embassy
APO San Francisco, CA 96503

Navy

1 SCIENTIFIC ADVISOR TO THE CHIEF
OF NAVAL PERSONNEL
NAVAL BUREAU OF PERSONNEL (PERS OR)
RM. 4410, ARLINGTON ANNEX
WASHINGTON, DC 20370

1 DR. RICHARD A. POLLAK
ACADEMIC COMPUTING CENTER
U.S. NAVAL ACADEMY
ANNAPOLIS, MD 21402

1 Mr. Arnold I. Rubinstein
Human Resources Program Manager
Naval Material Command (0344)
Room 1044, Crystal Plaza #5
Washington, DC 20360

1 Dr. Worth Scanland
Chief of Naval Education and Training
Code N-5
NAS, Pensacola, FL 32508

1 A. A. SJOHOLM
TECH. SUPPORT, CODE 201
NAVY PERSONNEL R& D CENTER
SAN DIEGO, CA 92152

1 Mr. Robert Smith
Office of Chief of Naval Operations
OP-987E
Washington, DC 20350

1 CDR Charles J. Theisen, JR. MSC, USN
Head Human Factors Engineering Div.
Naval Air Development Center
Warminster, PA 18974

1 W. Gary Thomson
Naval Ocean Systems Center
Code 7132
San Diego, CA 92152

Army

1 DR. JAMES BAKER
U.S. ARMY RESEARCH INSTITUTE
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333

1 DR. RALPH DUSEK
U.S. ARMY RESEARCH INSTITUTE
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333

1 DR. FRANK J. HARRIS
U.S. ARMY RESEARCH INSTITUTE
5001 EISENHOWER AVENUE
ALEXANDRIA, VA 22333

1 Dr. Milton S. Katz
Individual Training & Skill
Evaluation Technical Area
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

1 Dr. J. E. Uhlaner
Chief Psychologist, US Army
Army Research Institute
5001 Hector Road
McLean, VA 22101

1 Dr. Joseph Ward
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Air Force

1 Air Force Human Resources Lab
AFHRL/PED
Brooks AFB, TX 78235

1 Air University Library
AUL/LSE 76/443
Maxwell AFB, AL 36112

1 DR. G. A. ECKSTRAND
AFHRL/AS
WRIGHT-PATTERSON AFB, OH 45433

1 Dr. Alfred R. Fregly
AFOSR/NL, Bldg. 410
Bolling AFB, DC 20332

1 CDR. MERCER
CNET LIAISON OFFICER
AFHRL/FLYING TRAINING DIV.
WILLIAMS AFB, AZ 35224

1 Dr. Ross L. Morgan (AFHRL/ASR)
Wright -Patterson AFB
Ohio 45433

1 Research Branch
AFMPC/DPMYP
Randolph AFB, TX 78148

1 Dr. Marty Rockway (AFHRL/TT)
Lowry AFB
Colorado 80230

1 Brian K. Waters, Maj., USAF
Chief, Instructional Tech. Branch
AFHRL
Lowry AFB, CO 80230

Marines

1 Director, Office of Manpower Utilization
HQ, Marine Corps (MPU)
BCB, Bldg. 2009
Quantico, VA 22134

1 DR. A.L. SLAFKOSKY
SCIENTIFIC ADVISOR (CODE RD-1)
HQ, U.S. MARINE CORPS
WASHINGTON, DC 20380

CoastGuard

1 MR. JOSEPH J. COWAN, CHIEF
PSYCHOLOGICAL RESEARCH (G-P-1/52)
U.S. COAST GUARD HQ
WASHINGTON, DC 20590

Other DoD

12 Defense Documentation Center
Cameron Station, Bldg. 5
Alexandria, VA 22314
Attn: TC

1 Military Assistant for Human Resources
Office of the Director of Defense
Research & Engineering
Room 3D129, the Pentagon
Washington, DC 20301

1 Dr. Harold F. O'Neil, Jr.
Advanced Research Projects Agency
Cybernetics Technology, Rm. 623
1400 Wilson Blvd.
Arlington, VA 22209

1 Director, Research & Data
OSD/MRA&L (Rm. 3B919)
The Pentagon
Washington, DC 20301

1 DR. ROBERT YOUNG
ADVANCED RESEARCH PROJECTS AGENCY
1400 WILSON BLVD.
ARLINGTON, VA 22209

Civil Govt

1 Dr. William Gorham, Director
Personnel R&D Center
U.S. Civil Service Commission
1900 E Street NW
Washington, DC 20415

1 Dr. Andrew R. Molnar
Science Education Dev.
and Research
National Science Foundation
Washington, DC 20550

1 Dr. Thomas G. Sticht
Basic Skills Program
National Institute of Education
1200 19th Street NW
Washington, DC 20208

1 Dr. Vern W. Urry
Personnel R&D Center
U.S. Civil Service Commission
1900 E Street NW
Washington, DC 20415

1 Dr. Joseph L. Young, Director
Memory & Cognitive Processes
National Science Foundation
Washington, DC 20550

Non Govt

1 PROF. EARL A. ALLUISI
DEPT. OF PSYCHOLOGY
CODE 287
OLD DOMINION UNIVERSITY
NORFOLK, VA 23508

1 Dr. John R. Anderson
Dept. of Psychology
Yale University
New Haven, CT 06520

1 DR. MICHAEL ATWOOD
SCIENCE APPLICATIONS INSTITUTE
40 DENVER TECH. CENTER WEST
7935 E. PRENTICE AVENUE
ENGLEWOOD, CO 80110

1 MR. SAMUEL BALL
EDUCATIONAL TESTING SERVICE
PRINCETON, NJ 08540

1 Dr. Gerald V. Barrett
Dept. of Psychology
University of Akron
Akron, OH 44325

Non Govt

1 Dr. Kenneth E. Clark
College of Arts & Sciences
University of Rochester
River Campus Station
Rochester, NY 14627

1 Dr. Norman Cliff
Dept. of Psychology
Univ. of So. California
University Park
Los Angeles, CA 90007

1 Dr. Allan M. Collins
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, Ma 02138

1 Dr. John J. Collins
Essex Corporation
201 N. Fairfax Street
Alexandria, VA 22314

1 Dr. Meredith Crawford
5605 Montgomery Street
Chevy Chase, MD 20815

1 Dr. Nicholas A. Bond
Dept. of Psychology
Sacramento State College
600 Jay Street
Sacramento, CA 95819

1 Dr. John Seeley Brown
Bolt Beranek & Newman, Inc.
50 Moulton Street
Cambridge, MA 02138

1 DR. C. VICTOR BUNDERSON
WICAT INC.
UNIVERSITY PLAZA, SUITE 10
1160 SO. STATE ST.
OREM, UT 84057

1 Dr. John Carroll
Psychometric Lab
Univ. of No. Carolina
Davie Hall 013A
Chapel Hill, NC 27514

Non Govt

1 Dr. Richard L. Ferguson
The American College Testing Program
P.O. Box 168
Iowa City, IA 52240

1 Dr. Victor Fields
Dept. of Psychology
Montgomery College
Rockville, MD 20850

1 Dr. Edwin A. Fleishman
Advanced Research Resources Organ.
8555 Sixteenth Street
Silver Spring, MD 20910

1 Dr. John R. Frederiksen
Bolt Beranek & Newman
50 Moulton Street
Cambridge, MA 02138

1 DR. ROBERT GLASER
LRDC
UNIVERSITY OF PITTSBURGH
3939 O'HARA STREET
PITTSBURGH, PA 15213

1 DR. JAMES G. GREENO
LRDC
UNIVERSITY OF PITTSBURGH
3939 O'HARA STREET
PITTSBURGH, PA 15213

1 Dr. Barbara Hayes-Roth
The Rand Corporation
1700 Main Street
Santa Monica, CA 90406

1 Dr. Donald Dansereau
Dept. of Psychology
Texas Christian University
Fort Worth, TX 76129

1 DR. RENE V. DAVIS
DEPT. OF PSYCHOLOGY
UNIV. OF MINNESOTA
75 E. RIVER RD.
MINNEAPOLIS, MN 55455

1 Dr. Ruth Day
Center for Advanced Study
in Behavioral Sciences
202 Junipero Serra Blvd.
Stanford, CA 94305

1 MAJOR I. N. EVONIC
CANADIAN FORCES PERS. APPLIED RESEARCH
1107 AVENUE ROAD
TORONTO, ONTARIO, CANADA

Non Govt

1 DR. LAWRENCE B. JOHNSON
LAWRENCE JOHNSON & ASSOC., INC.
SUITE 502
2001 S STREET NW
WASHINGTON, DC 20009

1 Dr. Arnold F. Kanarick
Honeywell, Inc.
2600 Ridgeway Pkwy
Minneapolis, MN 55413

1 Dr. Roger A. Kaufman
203 Dodd Hall
Florida State Univ.
Tallahassee, FL 32306

1 Dr. Steven W. Keele
Dept. of Psychology
University of Oregon
Eugene, OR 97403

1 LCOL. C.R.J. LAFLEUR
PERSONNEL APPLIED RESEARCH
NATIONAL DEFENSE HQS
101 COLONEL BY DRIVE
OTTAWA, CANADA K1A 0K2

1 Dr. Robert R. Mackie
Human Factors Research, Inc.
6780 Cortona Drive
Santa Barbara Research Pk.
Goleta, CA 93017

1 Dr. William C. Mann
USC-Information Sciences Inst.
4676 Admiralty Way
Marina del Rey, CA 90291

1 Library
HumRRO/Western Division
27357 Berwick Drive
Carmel, CA 93921

1 Dr. Earl Hunt
Dept. of Psychology
University of Washington
Seattle, WA 98105

1 Dr. Richard B. Millward
Dept. of Psychology
Hunter Lab.
Brown University
Providence, RI 82912

1 Dr. Donald A Norman
Dept. of Psychology C-009
Univ. of California, San Diego
La Jolla, CA 92093

Non Govt

1 Dr. Jesse Orlansky
Institute for Defense Analysis
400 Army Navy Drive
Arlington, VA 22202

1 MR. LUIGI PETRULLO
2431 N. EDGEWOOD STREET
ARLINGTON, VA 22207

1 DR. PETER POLSON
DEPT. OF PSYCHOLOGY
UNIVERSITY OF COLORADO
BOULDER, CO 80302

1 DR. DIANE M. RAMSEY-KLEE
R-K RESEARCH & SYSTEM DESIGN
3947 RIDGEMONT DRIVE
MALIBU, CA 90265

1 Dr. Mark D. Reckase
Educational Psychology Dept.
University of Missouri-Columbia
12 Hill Hall
Columbia, MO 65201

1 Dr. Joseph W. Rigney
Univ. of So. California
Behavioral Technology Labs
3717 South Hope Street
Los Angeles, CA 90007

1 Dr. Andrew M. Rose
American Institutes for Research
1055 Thomas Jefferson St. NW
Washington, DC 20007

1 Dr. Leonard L. Rosenbaum, Chairman
Department of Psychology
Montgomery College
Rockville, MD 20850

1 DR. WALTER SCHNEIDER
DEPT. OF PSYCHOLOGY
UNIVERSITY OF ILLINOIS
CHAMPAIGN, IL 61820

Non Govt

1 DR. ROBERT J. SEIDEL
INSTRUCTIONAL TECHNOLOGY GROUP
HUMRRO
300 N. WASHINGTON ST.
ALEXANDRIA, VA 22314

1 Dr. Richard Snow
School of Education
Stanford University
Stanford, CA 94305

1 Dr. Robert Sternberg
Dept. of Psychology
Yale University
Box 11A, Yale Station
New Haven, CT 06520

1 DR. ALBERT STEVENS
BOLT BERANEK & NEWMAN, INC.
50 MOULTON STREET
CAMBRIDGE, MA 02138

1 Mr. D. J. Sullivan
c/o Canyon Research Group, Inc.
741 Lakefield Road
Westlake Village, CA 91361

1 DR. PATRICK SUPPES
INSTITUTE FOR MATHEMATICAL STUDIES IN
THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CA 94305

1 Dr. Kikumi Tatsuoka
Computer Based Education Research
Laboratory
252 Engineering Research Laboratory
University of Illinois
Urbana, IL 61801

1 DR. PERRY THORNDYKE
THE RAND CORPORATION
1700 MAIN STREET
SANTA MONICA, CA 90406

1 Dr. Benton J. Underwood
Dept. of Psychology
Northwestern University
Evanston, IL 60201

Non Govt

1 DR. THOMAS WALLSTEN
PSYCHOMETRIC LABORATORY
DAVIE HALL 013A
UNIVERSITY OF NORTH CAROLINA
CHAPEL HILL, NC 27514

1 Dr. Claire E. Weinstein
Educational Psychology Dept.
Univ. of Texas at Austin
Austin, TX 78712

1 Dr. David J. Weiss
N660 Elliott Hall
University of Minnesota
75 E. River Road
Minneapolis, MN 55455

1 DR. KEITH WESCOURT
INSTITUTE FOR MATHEMATICAL STUDIES IN
THE SOCIAL SCIENCES
STANFORD UNIVERSITY
STANFORD, CA 94305

1 DR. SUSAN E. WHITELY
PSYCHOLOGY DEPARTMENT
UNIVERSITY OF KANSAS
LAWRENCE, KANSAS 66044